I-centric Communications

vorgelegt von
Diplom-Informatiker
Stefan Arbanowski
aus Berlin

von der Fakultät IV – Elektrotechnik Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Promotionsausschuß:

Vorsitzender: Prof. Dr.-Ing. Adam Wolisz
Berichter: Prof. Dr. Dr. h.c. Radu Popescu-Zeletin
Berichter: Prof. Dr. Bernd Mahr

Tag der wissenschaftlichen Aussprache: 21. November 2003

Berlin 2003
D 83

I-centric Communications

vorgelegt von
Diplom-Informatiker
Stefan Arbanowski
aus Berlin

von der Fakultät IV – Elektrotechnik Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades

Doktor der Ingenieurwissenschaften
– Dr.-Ing. –

genehmigte Dissertation

Berlin 2003

Abstract

This thesis describes the vision of **I-centric communications** – a new paradigm for future telecommunication systems. The main objective of this approach is to consider the human communication behavior, not the technologies that support communication, as the starting point for the design of telecommunication systems. The focus of this thesis is on the conceptual integration of all aspects of I-centric communications.

Looking at the communication behavior of human, it is obvious, that human beings frequently interact with a set of objects in their environment. Following this view, a new approach is not to build communication systems based on specific technologies, but on the analysis of the individual communication space.

The result is a communication system that adapts to the demands of each individual (I-centric). The communication system will act on behalf of human's demands, reflecting recent actions to enable self-adaptation. I-centric Services adapt to individual communication spaces and situations. In this context 'I' means I, or individual, 'Centric' means adaptable to I requirements and the individual's environment.

The rationales above require intelligence in service provisioning in order to personalize, adapt to situational and environmental conditions, to monitor and to control the individual communication space. An I-centric communications system will provide the intelligence required for modeling the communication space of each individual adapting to its interests, environment, and preferences.

The thesis introduces the vision of I-centric communications, followed by the development of a reference model for I-centric communications. Since both, the vision and the reference model, are general, an architectural framework for I-centric communications is introduced later on. This framework is used to design and implement an I-centric communications system.

The work in the area of I-centric communications was carried out in cooperation of the Department for Open Communication Systems (OKS) at the Technical University Berlin (TUB) and the Fraunhofer Institute FOKUS. The vision and the reference model, introduced in this thesis, are the results of this cooperation.

The main research directions for the cooperation between TUB and FOKUS have been a general model for I-centric Services, the service platform for I-centric Services, and an approach for the interaction of users with I-centric Services.

This thesis focuses on I-centric Services and according communication system. The aspects of user interaction are out of scope of this thesis. Nevertheless, these aspects have been analyzed by Stephan Steglich, researcher at the TUB, in a second PhD thesis in parallel.

The results of this thesis have been contributed to different national and international projects (BMBF: LiveFutura, VHE, PI-AVIda, IST: WSI, WWRI), standardization bodies (OMG, WWRF), conference papers, and journals by introducing the vision of I-centric communications to a larger auditorium, and by exploiting parts of the developed I-centric systems.

German Abstract

Diese Arbeit beschreibt die Vision „I-centric Communications“ – ein neues Paradigma für zukünftige Telekommunikationssysteme. Das Hauptziel dieses Ansatzes ist es das menschliche Kommunikationsverhalten und nicht die Technologien die typischerweise zur Kommunikation eingesetzt werden, als Ausgangsbasis für den Entwurf von Telekommunikationssystemen zu benutzen. Daher konzentriert sich diese Arbeit auf die konzeptionelle Integration aller für I-centric Communications notwendigen Aspekte.

Untersucht man das Kommunikationsverhalten von Menschen näher, wird klar, dass Menschen regelmäßig mit einer bestimmten Menge von Objekten in ihrer Umgebung interagieren um ihre täglichen Probleme zu lösen. Diese Erkenntnis kann als Grundlage für den Entwurf von Telekommunikationssystemen dienen. Der Entwurf stützt sich dabei nicht auf spezifische Telekommunikationstechnologien sondern auf die Analyse individueller Kommunikationsräume.

Das Resultat eines solchen Entwurfs ist ein Telekommunikationssystem, dass sich an die individuellen Anforderungen eines jeden Menschen anpasst (I-centric). Das Kommunikationssystem agiert im Sinne des Benutzers, protokolliert wiederkehrende Aktionen, um sich selbst den Bedürfnissen des Benutzers weiter anzupassen.

Die beschriebene Funktionalität erfordert eine intelligente Diensterbringung im Hinblick auf die Personalisierung von Diensten, der Adaption an situationsbedingte oder umgebungsabhängige Einflüsse, und der Überwachung bzw. Steuerung von individuellen Kommunikationsräumen. Kommunikationssysteme die dieser Idee folgen müssen individuelle Kommunikationsräume modellieren und verwalten um sich an die persönlichen Interessen, Umgebungen und Präferenzen anpassen zu können.

Diese Arbeit beginnt mit der Vorstellung der Vision „I-centric Communications“, gefolgt von der Entwicklung eines entsprechenden Referenzmodells. Im Anschluss wird auf der Basis des Referenzmodells eine Architektur zur Implementierung von I-centric Kommunikationssystemen entwickelt. Die Vorstellung einer der Architektur folgenden Implementierung schließt diese Arbeit ab.

Die Forschungsarbeit im Bereich „I-centric Communications“ wurde in Kooperation zwischen dem Lehrstuhl für Offene Kommunikationssysteme (OKS) der Technischen Universität Berlin (TUB) und dem Fraunhofer Institut FOKUS durchgeführt. Die Vision sowie das Referenzmodell für „I-centric Communications“, die in der vorliegenden Arbeit vorgestellt werden, sind Ergebnisse dieser Kooperation.

Die Forschungsschwerpunkte der Kooperation zwischen TUB und FOKUS waren das „Generelle Modell für I-centric Dienste“, die „Dienstplattform für I-centric Dienste“ sowie die „Interaktion zwischen Nutzern und I-centric Diensten“. Die vorliegende Arbeit konzentriert sich auf das generische Model für I-centric Dienste und entsprechende Dienstplattformen. Aspekte der individuellen Benutzerinteraktion werden nicht betrachtet. Diese Aspekte wurden in einer zweiten Dissertation von Stephan Steglich, TUB, analysiert und ausgearbeitet.

Die Ergebnisse dieser Dissertation wurden in verschiedenen nationalen und internationalen Forschungsprojekten (BMBF: LiveFutura, VHE-UD, PI-AVIda, IST: WSI, WWRI), Standardisierungsgremien (OMG, WWRF), Konferenzpapieren sowie Zeitschriften eingebracht, um die Vision von „I-centric Communications“ einem größeren Auditorium vorzustellen.

Acknowledgements

I extend my gratitude to Radu Popescu-Zeletin. He gave me the opportunity to work in both of his unique environments; the department of Open Communication Systems, OKS (at the Technical University of Berlin), and the Fraunhofer Society' Research Institute for Open Communication Systems (Fraunhofer FOKUS). I like to thank him for his strategic visions, an uncountable number of discussions about I-centric communications, and his continuous support during all phases of this thesis.

Further, my gratitude goes to Stephan Steglich for our inspiring debates, his constructive criticism, and for offering varying perspectives on many issues, related, and unrelated to I-centric communications.

I express my warm thanks to all the university students graduating with their work on the I-centric communications system. Their contribution made it possible for this thesis to be written. Among them, I thank Holger Waterstrat for his exceptional degree of creativity during his work.

Of course, I thank all former and current colleagues in the above institutes, providing the technical support and social environment for everyday life and research.

Last, but not least, I give my heartiest gratitude and love to my girlfriend Katja and to my parents Marlie and Dieter Arbanowski. Thanks for your patience.

Table of Content

1 Introduction

In 2000, the IST Programme Advisory Group (ISTAG) published a vision statement [ISTAG01] for Framework Programme 5 that laid down a challenge to:

'Start creating an ambient intelligence landscape (for seamless delivery of services and applications) in Europe relying also upon test-beds and open source software, develop user-friendliness, and develop and converge the networking infrastructure in Europe to world-class'.

Ambient Intelligence stems from the convergence of three key technologies: Ubiquitous Computing, Ubiquitous Communication, and Intelligent User Friendly Interfaces. According to the vision statement, on convergence humans will be surrounded by intelligent interfaces supported by computing and networking technology, which is embedded everywhere in everyday objects such as furniture, clothes, vehicles, roads, and smart materials even particles of decorative substances like paint. Ambient Intelligence implies a seamless environment of computing, advanced networking technology, and specific interfaces. It is aware of the specific characteristics of human presence and personalities, takes care of needs and is capable of responding intelligently to spoken or gestured indications of desire, and even can engage in intelligent dialogue. Ambient Intelligence should also be unobtrusive, often invisible: everywhere and yet in our consciousness – nowhere unless we need it. Interaction should be relaxing and enjoyable for the citizen, and not involve a steep learning curve. [ISTAG01]

Figure 1: Individual communication space

In the last years, a variety of concepts for service integration and corresponding systems has gained momentum. On one hand, they aim at the interworking and integration of classical telecommunications and data communications services, such as telephony, voicemail, fax, e-mail, paging, etc. [Emm97, Hov97] On the other hand, they focus on universal service access from a variety of end user systems, including both fixed and mobile terminals. [Arb99b, Coen91] All these systems are driven by the concept of providing several technologies to users by keeping the peculiarity of each service.

The prerequisite for service integration is global connectivity, based on a fast developing web of interconnected communication networks, comprising both fixed and wireless networks. In addition, the provision of a global service infrastructure, based on network-independent open service platforms is the other fundamental prerequisite, hiding the complexity of network diversity, and allowing the fast and efficient creation, provision, and management of future services.

The spectrum of services ranges from simple communication services up to complex distributed applications. In particular, new intelligent information and messaging services are common-

place. [Weiser91] Additionally, services providing information retrieval (news on demand, brokerage information on demand) and services emerging from the field of electronic commerce (hotel or flight reservation) or facility control (intelligent home) are gaining momentum.

This requires appropriate communication services and information systems that adapt automatically to the human communication behavior. Users have to be supported in order to cope with the huge amount of information available and the increasing complexity of (communication) services. In addition, the continuous reachability of people due to advanced mobile communication services requires adequate means for information filtering and communication control.

1.1 Motivation

It is common understanding that future services will have to adapt to individual requirements of human beings [BoV01, Nor93]. The communication system has to provide the intelligence required for modeling the communication space of each individual adapting to its interests, environment, and life stage.

I-centric communications considers the human behavior as a starting point to adapt the activities of communication systems to it. Human beings do not want to employ technology. As Figure 1 shows, humans rather want to communicate interacting in their **individual communication space**.

Introducing communication systems that follow the I-centric vision can only be an evolutionary process. A transition phase between 3rd generation telecommunication networks[1] towards 3G beyond (3G*b*) systems is needed. 3G*b* is anticipated as enabling platform for user-driven service provisioning incorporating a variety of access mechanisms, mobile service execution environments, and a huge amount of mobile services [Arb02b]. I-centric concepts will help to realize the transition by providing the needed methodology and by selecting enabling technologies, which are partially available already.

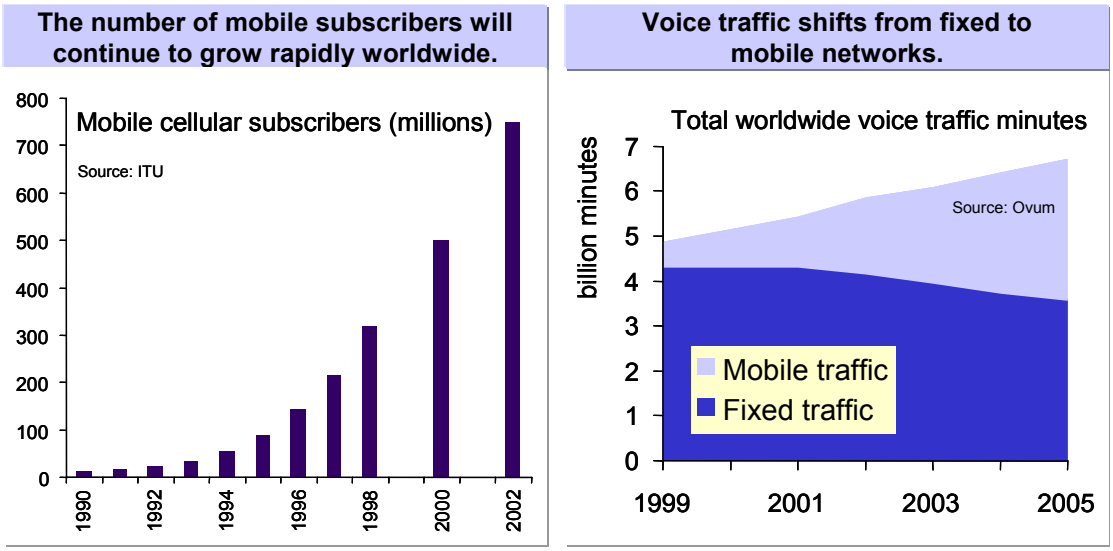


Figure 2: Communication via mobile networks

Significant technological advances in recent years in the areas of palm-sized computers and wireless communications, accompanied by an infiltration of the Internet in all aspects of our live have given reason to analysts world-wide to forecast that the Internet population will consist more and more of mobile access devices (see Figure 2).

[1] also known as IMT-2000 (ITU/ETSI), UMTS (3GPP), or 3G for 3rd generation mobile networks

A variety of different wireless network technologies capable, of transporting Internet traffic, will become available. IP-based technologies that allow the integration of available heterogeneous networks into a single platform capable of supporting user to roam between them, while not interrupting active communications have already been introduced [W-FSAN]. This development will be assisted by the rise of new mobile devices maintaining various access interfaces that will allow connectivity over a range of providers and technologies. Finally, the emergence of a variety of access devices will dictate the liberation of users from a single device and allow service mobility between devices and networks [Mohr02, Arb03a].

1.2 Trends towards 3Gb Telecommunication Systems

The driving forces for today's telecommunication world have been the growth of Internet (especially broadband internet access at home) and 2G[2] mobile telecommunication. Both are now converging around the 2.5[3] and 3G[4] networks and services. The telecommunications industry is characterized by complex business models caused by the following drivers:

- services realized by the integration and co-operation of services from a number of players (Network Operators, Internet Service Providers, Content Providers, etc.),
- co-existence of Internet based and traditional telecommunication services,
- the ongoing integration of pure data, voice, video and value-added services into a single service stream,
- demand for mobility with unified service access offering guaranteed QoS, and
- cost effective delivery of service (irrespective of access technology).

From a service perspective, the main difference between 2G/2.5G and 3G telecommunication systems is the new air interface enabling higher transmission data rates. For 3G beyond systems a global consensus exists that a new system architecture needs to be developed. This system architecture has to support a number of new features that have been identified already:

- blurring business roles (nearly everyone can be provider, operator, customer, and user)
- personalized, ambient-aware, adaptive end user services
- augmented environments as part of the ubiquitous communication system (SmartIP devices and sensor networks [Pfe03])
- new networking services: ad-hoc, p2p (content aware, secure, QoS aware)
- all IP services: always best connected, packet switched, broadband multimedia applications
- flexible platform supporting diverse access technologies, global coverage, global roaming
- further convergence of voice, data, and mobile communications
- new wireless links (high/low data rate, long/short range) to serve different application domains

Beside all these technical trends, a harmonization between different application domains is expected. The integration between services for office and home environments, which has been started already, will continue towards service environments covering the complete communication space of individual users.

As shown in Figure 3 telecommunication 3G beyond systems will support mobile service usage irrespective where users are, what kind of terminal they are using, what kind of bearer technology is underneath, and what kind of information has to be delivered.

[2] 2G: Global System for Mobile Communications (GSM) – 9,6 kBit/s

[3] 2.5G: GSM+General Packet Radio Service (GPRS) and High Speed Circuit Switched Data (HSCSD) – typically 53,6 kBit/s up to 384kBit/s

[4] 3G: Universal Mobile Telecommunication System (UMTS) – typically 128 kBit/s up to 384kBit/s

1.2.1 Telecommunication and Wireless Networks

Started in the mid 90th, telecom markets are still characterized by the convergence of traditional telecommunication services (e.g. telephony) and Internet / packed based services (e.g. WWW, Email). Many of these services are available today without a harmonized appearance towards the user.

Service provisioning and especially personalization of services is perceived as an upcoming and important success factor. Today, elements of personalization are already present to some extent, even though not in areas specifically related to the service provision (e.g., ringing tones that can be downloaded on the handset, covers of different colors that can be bought for a number of phones, etc.). The current success of these gadgets and basic features has proven the user acceptance of personalized communication environments already.

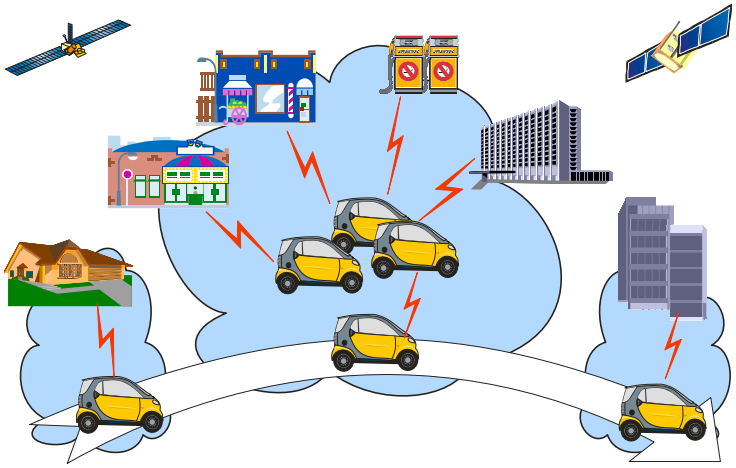


Figure 3: Mobile Service Usage

Although, some services offer the possibility of personalization, the support of individual user needs is proprietary, at best. In the World Wide Web, typical ways of personalization are to monitor the user's click-stream and to observe his behavior in order to learn something about his interests. As no common structure or format is available to describe and store this information, the service provider keeps it locally and does not provide this information to other service providers or to the users themselves.

Market analysis has already required future services to be user-centric. They should adapt to user needs, and to the current situation, the user is in. Traditional communication services, designed for large user groups, are not able to address individual user needs.

The concept of ambient intelligence has been introduced in 2000 by ISTAG [ISTAG]. Provision of ambient-aware services is the next frontier towards the realization of such concept. Location based techniques are the first to be exploited in this direction, and some operators are offering location-based services in GSM and UMTS already.

Another trend is the introduction of service and content adaptation. In the area of content adaptation, technologies have been developed that allow description and transformation of content in such a way that it can be presented by devices with different characteristics (e.g. size of the screen, graphic capabilities of the screen, etc.). Research has also been performed on coding algorithms that allow information to be trans-coded in relation to e.g. the current network status. In addition, media conversion is an available feature today and it has been introduced in commercial Unified Messaging Systems already [Arb99a, W-UMS].

Some of the main architectural breakthroughs in respect to a general service architecture have been developed by the TINA[5] consortium [W-TINA-SA]. The main results are the definition of

[5] Telecommunication Information Networking Architecture (TINA) [W-TINA-C]

a telecommunication related business model, the definition of an associated session model for single-party and multi-party services and the specification of related APIs, called reference points, to guarantee interoperability between different implementations, provided by different service providers. Additionally a CORBA-based[6], proprietary run-time system called distributed processing environment (DPE) has been specified. Even though TINA introduced the notion of profiles to store required service characteristics, TINA does not provide solutions for user-centric or ubiquitous computing. The TINA business model implies a centralized architecture, where components are assigned to fixed domains. In a mobile environment as envisioned for 3G beyond, this approach and the business model itself have to be extended. TINA has told many lessons to be considered, but the general ideas have to be extended towards user-centric, ubiquitous, and mobile environments.

1.2.2 Internet Protocol Everywhere

The era of monolithic telecommunication networks with centralized intelligence (like Intelligent Networks [IN, Mag96]) is developing towards decentralized structures where the borderline between the traditional roles of network provider, content provider, service provider, and user vanishes. [Win98] This is due to the penetration of network technologies everywhere and by everyone (sees Figure 4). The integration of these networks pertaining to even different administrative domains is solved on OSI[7] layer 3 by using the Internet protocol (IP).

Figure 4: Network Penetration

Another major trend is the proliferation of IP-based devices. End-systems will not only be desktop/laptop computers but also Smart-IP devices. All these devices will be addressable in a global IP communication infrastructure connected via various wired and wireless networks.

Considering these facts, it is obvious that any microcontroller-based device has the potential to be part of the communication space in the future. Devising service architecture for such environment is fundamental for future communication systems.

Present communication systems are designed and developed for specific end-systems and for a specific communication service (e.g. Fax for facsimile, TV broadcasting for TV sets, and telephony for telephone sets). A vertical design from network technology up to the user interface and device capabilities takes place. Services are kept presentation oriented and each of them has its own way to handle it. Every service is developed for a certain network technology, dedicated to a user community in which individuals are reduced to the common denominator defined by the service designer. In this sense, services are generic without limitations. The hope is that everybody will buy the service and the associated device. This implies the communication infrastructure to be engineered to offer the broadest solution. Scalability, performance, and controllability of the network infrastructure are the resulting problems.

The progress of high-performance and low-cost processor technology is enabling computer power to be embedded densely in devices like mobile phones, PDAs, and facilities like Internet

[6] Common Object Request Broker Architecture (CORBA) [W-CORBA]

[7] Open System Interconnection (OSI) [W-OSI]

appliances as well as desktop computers. They are most likely to become present everywhere: at home and in the office, on the street, in cars, in factories, etc. Wireless networks like Bluetooth, WLAN, or ZigBee [W-ZigBee] enable these devices to be connected to each other without new wires, and IPv6 supports their interconnection by accommodating a massive number of devices. Available computer power and the networks lead to the ubiquitous access to services – anytime, anywhere, and for anyone. Furthermore, such networks will enhance safety, security, and quality/comfort of our live.

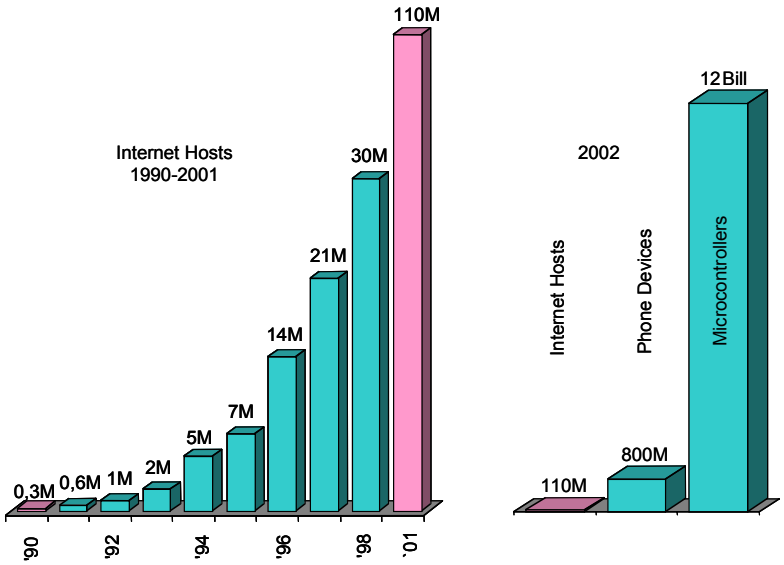


Figure 5: Proliferation of IP-based Devices

Much technical and business attention is paid to new computing paradigms like mobile computing, ubiquitous computing, and pervasive computing[8] driven by these technical trends. A telecommunication system, using IP enabled devices, is constructed through the connection to already existing networks, namely the Internet. Although there are standards for interconnecting devices like HAVi [W-HAVi], UPnP [W-UPnP], JXTA [W-JXTA], and Jini [W-Jini] in different application domain, no common standards exist to handle these devices for various service applications, across these domains, in a unified manner.

The idea for Super Distributed Objects (SDO) [SDO-WP] is to provide a standard computing infrastructure that models real world entities (e.g. devices) as objects, deploys them in a highly distributed environment, allows them to seamlessly interwork with each other, and ubiquitously aids users in accomplishing their tasks. Super distribution means incorporating massive numbers of objects beyond centralized control, each of which performs its task autonomously or cooperatively with other objects. Incorporating these characteristics in a distributed object system requires addressing issues like ad-hoc interaction between objects, temporary unavailability of objects, and peer-to-peer computing.

1.2.3 The Convergence of these Worlds

Taking emerging 3G business models and value networks into account, services in 3G beyond environments are expected to be implemented in concert of many actors using open and standardized interfaces. The 3GPP[9] promotes the possibility to create services on standardized tool sets, e.g. OSA[10]. OSA defines an architecture that enables operator and third party applications to make use of network functionality through an open standardized API. It offers the liaison

[8] For more details on pervasive computing, see the background of ambient-awareness (section 3.2.5.1).

[9] 3rd Generation Partnership Project (3GPP) [W-3GPP]

[10] Open Service Access (OSA) [W-OSA]

between applications and service capabilities provided by the network. In this way, network complexity is transparent to the applications that are presented independent of the underlying network technology. Closely related to OSA is the work of the Parlay group[11], which has developed an open, technology-independent, and extensible service API for any type of bearer network.

The Virtual Home Environment presents the user with a common 'look and feel' and service experience regardless of location, network, and terminal type. The VHE is based on standardized service capabilities and personalized features that are consistently presented so that the user always 'feels' that he is on his home network even when roaming across network boundaries. Full security will be provided transparently across a mix of access and core networks. [W-3GPP]

Another standardization activity in this area is carried out by OMA[12]. This alliance is focusing on an open standard based framework to permit services to be build, deployed, and managed efficiently in a multi-vendor environment. It tries to establish a mobile industry standard forum for creating service level interoperability.

The vision of information at any time, at any place, in any form has become popular in the mid 90th. Several research activities have investigated in that vision [Dec97]. In the last years, also the market has adapted that slogan to promote its offers. It has already been shown in [Pfe99], that it is possible to deliver any kind of information to any kind of terminal. To enable such an application, conversion processes have to be integrated in communication systems. These conversion processes could be used for the automatic service delivery and for the remote access to stored messages. In both cases, the conversion processes adapt a certain kind of message to the terminal of the user.

1.3 Structure of this Thesis

Following the vision of Ambient Intelligence, the department for Open Communication Systems at the Technical University Berlin has coined the term I-centric Communications. Chapter 2 provides the terminology defined for I-centric communications. It introduces the concept of I-centric communications and describes an according reference model, which has been used to design and to implement an I-centric Communications System.

Chapter 3 introduces the architectural framework for I-centric communications systems. The main components and their interactions are described to set the ground for ambient-aware, personalized, and adaptive services. A special emphasis is given to a model to describe and handle information that is needed for I-centric communications. Based on this model, the architectural framework supports advanced functions for user initiated services creation and distributed services execution on top of heterogeneous network environments.

The detailed modeling of the realized I-centric communications system, the engineering issues, the specific implementation, and integration of components are described in chapter 4, including the practical realization of an Open Profiling Framework and Super Distributed Objects.

Chapter 5 provides a summary and an outlook on work, which is already in progress and possible in future to enhance I-centric communications in general, and the implemented I-centric communications system especially.

References of related literature, a list of acronyms, and an appendix containing some technical details close this thesis.

[11] Parlay Group [W-Parlay]

[12] Open Mobile Alliance (OMA) [W-OMA]

2 I-centric Communications

Following the vision of Ambient Intelligence, the Department for Open Communication Systems at the Technical University Berlin has coined the term I-centric Communications, which picks up the idea of providing services based on the individual communication behavior of human beings. This chapter provides the terminology defined for this thesis. It introduces the concept of I-centric communications and provides an according reference model, which has been used to design and to implement an I-centric Communications System.

The work in the area of I-centric communications was carried out in cooperation of the Department for Open Communication Systems (OKS) at the Technical University Berlin (TUB) and the Fraunhofer Institute FOKUS. The vision and the reference model, introduced in this chapter, are the results of this cooperation. The results have already been contributed to several national and international projects, conference papers, journals, and diploma theses.

The main research directions for this cooperation have been:

- *a general model for I-centric Services,*
- *the service platform for I-centric Services, and*
- *an approach for the interaction of users with I-centric Services.*

This thesis focuses on I-centric Services and according service platform. The aspects of user interaction are out of scope of this thesis. Nevertheless, these aspects have been analyzed by Stephan Steglich, researcher at the TUB, in a second PhD thesis[13] in parallel. The following sections (section 2.1-2.4) represent the basic framework for both PhD theses. They have been developed together and therefore are part of both PhD theses.

In this sense, both theses are complementary. They are based on the same vision, focusing on different aspects of I-centric communications. Section 2.5 introduces basis of this thesis. It relates the research tasks, which are unique for this thesis, to the vision of I-centric communications. Chapter 3 proposes an architectural framework for the realization of I-centric Services and according service platform. In chapter 4, an implementation of the introduced concept is described. It represents a first prototype of an I-centric communications system. This system was developed at Fraunhofer FOKUS. Due to the cooperation with TUB, first steps towards an integration of both works, the I-centric user interaction and the I-centric Service platform, have been carried out together.

2.1 Vision

The communication behavior of human beings is characterized by frequent interactions with a set of **objects** in their environment. Humans solve the problems of their daily life, e.g. money and bank accounts need to be managed, food has to be bought and to be prepared for eating, movies are watched for entertainment, places are visited and news are consumed to improve education, and other people are met for discussions. The set of objects, controlled by each individual human, define its **individual communication space** as shown in Figure 6.

A communication space of an individual is limited: 'I do not know everybody in the world, I am not interested in everything, and I do not have all necessary devices required by all communication services everywhere at all times'.

Furthermore, individuals are interested in semantic and not necessarily in the kind of presentation of a specific service. Services in an individual communication space have to support the quality of the human senses, and since quality of senses is individual, they have to **adapt** their presentation to each individual automatically. Services have to adapt to the life stage and the environment of each individual.

---

[13] Steglich, St.: '*I-centric User Interaction*', PhD thesis, Technical University Berlin, Falkultät IV, 2003

In the former days, the communication space of human beings has been limited to the actual surrounding physical environment (like a home or an office) because of the spatial range of human senses. With the introduction of telephony, the range was expanded. It became possible to talk with people regardless their location. With asynchronous services like email and Short Message Service (SMS), the dimension of time expanded. People can send emails and do not need to care, whether the addressee is ready to receive the message or not. That is, technology has eliminated distances in time and space or at least made these boundaries almost unperceivable. By this means, today's communication services act as a prolongation of human senses and extend the individual communication space.

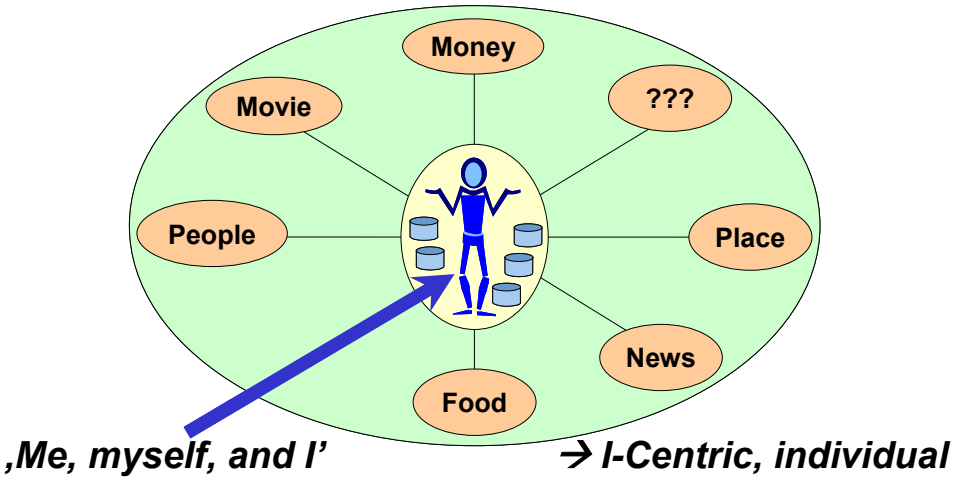


Figure 6: Individual communication space

Following this view, a new approach is to build communication systems not based on specific technologies, but on the analysis of the **individual communication space**. The result is a communication system that adapts to the specific demands of each individual (**I-centric**). Such a communication system will act on behalf of human's demands, reflecting recent actions to enable profiling, and self-adaptation. I-centric Services adapt to individual communication spaces and situations. In this context 'I' means I, or individual, 'Centric' means adaptable to I demands and a certain environment. [Arb00a]

The rationales above require intelligence in service provisioning in order to personalize, adapt to situational and environmental conditions, to monitor and to control the individual communication space. An I-centric communications system will provide the intelligence, which is required for modeling the individual communication space of each individual by adapting to its interests, environment, and preferences.

The multitude of devices, wearables, different telecommunication technologies, positioning and sensing systems are considered as enabling technologies for I-centric communications. Universal information access (including service interworking, media conversion), flexible management of equipment and facilities (e.g. smart homes [Well93a]), and personal communications (supporting personal mobility and terminal mobility [Eck96]) form the basis of such systems.

2.2 I-centric Communications – Basic Terminology

I-centric means to take a bottomless look at human behavior and to adapt the activities of communication systems to it. Human beings do not want to employ technology. They rather want to communicate acting in their individual **communication space**. They meet with others to talk, to celebrate, they read and travel, they are listening to news or to music, they take decisions, etc. This abstract description of humans' communication activities requires a set of definitions that allow the mapping of abstract requirements (or wishes) to the physical communication environment later on. In the following, the basic terminology to describe I-centric communications systems is introduced.

As discussed above, the human communication behavior is characterized by frequent interaction with a set of other humans, information sources, and devices within or outside humans' vicinity. All entities, humans are interacting with, will be called **objects** further on. They can be activated or deactivated[14] by an individual, or environmental conditions, to perform an action according to specific needs of an individual. Objects are directly addressable, and can represent one or more physical entities performing a certain service.

Object: An object is a logical representation of hardware or software entity, or even a representation of a certain individual, and provides well-defined services from the perspective of an (other) individual.

To model the interaction of human beings with objects of their individual communication space a general model applicable for all kinds of objects is needed. Objects will be used differently in different contexts by different individuals. In addition, a mechanism is needed for managing the use of objects (e.g. monitoring the activities of each object is used to profile, or to account service provisioning). The focus regarding individual-object interaction is to provide services of objects in a generic way.

To enable ad-hoc interaction of beforehand unrelated objects, an interaction model between objects is needed. This will allow the dynamic collaboration of objects for a specific purpose. Together with an organizational model, which describes relations between objects like ownership issues, such kind of interaction can be used to stimulate social behavior of objects, like multi-agent systems [Martin99, Arb98, and Bre98] do, to perform a specific task.

The basic idea of objects, to model real-world entities, has to be reflected by the I-centric Service architecture. General procedures for wrapping legacy technology have to be developed facing the fact that environmental constraints can affect the design of a distributed system [ODP, X.901-904]. Middleware concepts have to be selected that on one hand hide the heterogeneity of physical resources (represented by objects of individual communication spaces), but on the other hand support vertical integration, if necessary.

A general methodology how to select/design objects of individual communication spaces has to be developed. An I-centric system should support massive numbers of objects and should be tolerant against object failures. The population of objects is always changing because they spontaneously enter/leave/roam the environment and hence the communication space. Already standardized mechanisms for naming, lifecycle, monitoring, fault tolerance etc. have to be taken into account to determine whether they suit the requirements of I-centric communications.

Due to changes in human being's daily live, the amount or the concrete instances of objects are changing over time. Nevertheless, the sum of objects, an individual might interact with form his **individual communication space**. Objects may pertain to different communication spaces. They can be controlled by individuals, other objects, or services. Individuals can directly ask for a service to be performed by an object, whereas environmental condition may influence the status of objects indirectly.

Communication between different individuals takes place by sharing objects of their communication spaces. In this case, objects representing communication facilities in the different communication spaces will be connected to establish a physical connection between two individuals. What kind of physical resources are used for the communication is decided dynamically and depends on individual preferences of involved parties, their available communication facilities, and additional ambient information. The process of how to select and activate objects and physical resources underneath is one of the main activities of an I-centric communications system and will be introduced in section 2.2.1. [Arb01g, Arb01g]

[14] Activation or deactivation in this sense is an abstraction of using an object. The 'Usage' can refer to a complex task involving multimodal user interaction or complex business / service logic inside or on the outsite of an object.

Individual communication spaces are growing and shrinking in the time axes based on the individual life stage, personal interests, working and living environments, and the availability of new kinds of telecommunication services and devices. Meeting new friends or entering a room, which provides certain telecommunication devices, enlarges the communication space, whereas leaving the same room will reduce the size of the communication space dynamically.

This process of growing/shrinking the communication space has to be supported by an I-centric communications system in two ways. On one hand, the system must perform this process automatically, but on the other hand, the individual must have the possibility to include and exclude objects explicitly.

Individual Communication Space: The individual communication space of a certain individual is defined by a set of objects this individual might want to interact with. The size of the individual communication space varies over time due to the appearance or disappearance of objects.

Each individual has only **one** individual communication space. It contains all objects this individual might want to perform requests on. Objects that pertain to individual communication spaces of different individuals must handle concurrent access from different individuals or must delegate the concurrency control to the I-centric communications system.

Although the individual communication space provides a repository of objects for an individual there is no partitioning of objects in the communication space itself. The concept of context, personalization, and ambient-awareness provide necessary mechanisms to structure, classify, group, and even to partition objects by several means, i.e. relations between objects to define dependencies, or relations between objects and individuals to define ownership issues or usage rights.

2.2.1 Context and Active Context

An individual communication space provides a set of objects, whose services an individual can use, to achieve its goals. In addition to that, the concept of the **context** provides the definition of relationships and causalities between different objects of an individual communication space and the individual. A context represents a 'universe of discourse' in an individual communication space. Individuals communicate with objects in their environment in a certain context.

A context can be seen as a metaphor for:

- which objects of an individual communication space, an individual wants to use in parallel,
- which objects have to be taken into account to fulfill a certain wish of the individual, or
- whether and how objects are affected by activities of other objects.

Objects may pertain to different contexts (even to contexts of different individuals), because individuals might want to have a certain object involved in different activities.

Context: A context represents a certain 'universe of discourse'. It defines relationships and causalities of an individual to and between particular numbers of objects of its communication space.

Contexts are independent from any concrete environment. If an individual wants to **act in a certain context** this context has to be activated. An **active context** defines the relationship of an individual to and between particular numbers of physical resources (see Figure 7) at a certain moment in time, in a certain environment. The activation and deactivation of a context should usually be done automatically just by analyzing individuals' activities, but an individual should also have the possibility to do this explicitly.

Activating a context means:

- the identification of objects that are required by the context,
- the evaluation of the relationships and causalities between objects defined by the context,

- the discovery of the actual vicinity of the individual to identify physical resources that provide the functionality required by the identified objects,
- the activation/configuration of these physical resources to perform the task required by the context.

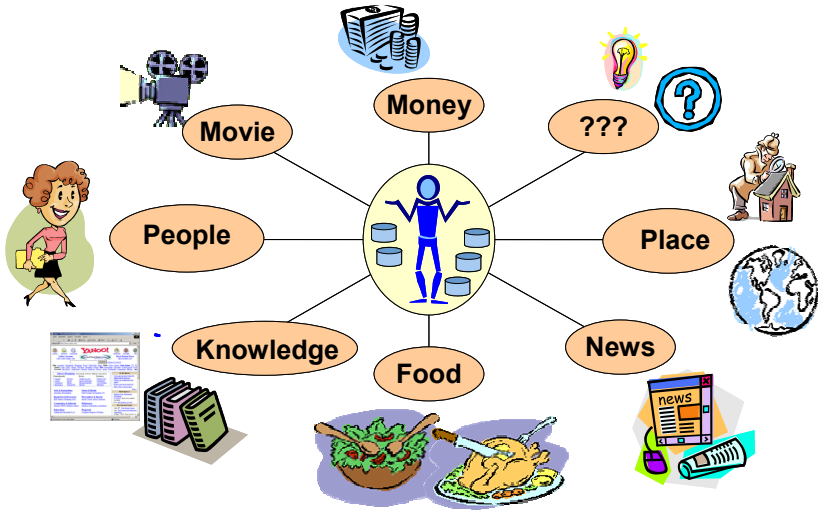


Figure 7: Individual communication space & underlying physical resources

The difference between **context** and **active context** is characterized by the entities, which are considered in relations and causalities. Context only refers to objects as an abstract model of what kind of objects have to be taken into account in a certain context, whereas an active context refers to physical resources that have been identified during the activation process. Active contexts are of dynamic nature reflecting the current environment an individual resides in.

The activation and deactivation of contexts is one task of I-centric Services. To activate a context the I-centric Service performs the activities described above. In addition, the I-centric communications system has to manage concurrent access to objects and conflicts caused by contrary wishes, expressed by individual(s).

Active Context: A context is active when it is adapted to a certain environment at a certain moment in time. It defines the relationships and causalities of an individual to a particular number of physical resources at certain a moment in time, in a certain environment.

Deactivating a context means:

- the identification of objects that have been activated/controlled in the context,
- the identification of affects that have been caused on other objects by the context,
- the identification of activities (e.g. deactivation, reconfiguration) that have to be performed on all these objects and physical resources underneath for deactivating the context,
- performing these activities on the objects and physical resources.

Acting in a context means to use only services that are provided by objects, which are part of that very context. Starting to interact with objects that are not part of an active context will cause the activation of another context.

That means, on one hand individuals are allowed to act in several contexts in parallel, and on the other hand, the I-centric communications system must handle conflicts that might occur due to contrary causalities defined in the different contexts.

To handle each individual communication space and associated contexts, a general model of domain information and relationships to objects and physical resources is needed. This model must be flexible to be enhanced due to the introduction of new locations, devices, etc. Furthermore, the model has to provide mechanisms to map objects to physical devices to support the activation and deactivation of contexts.

2.2.2 Preferences and Ambient Information

Individuals have different **preferences** in different situations. Sitting alone in a silent room might indicate that an individual is willing to receive incoming phone calls. However, the same individual can perceive this as a disturbance when being involved in a conversation with others. With preferences, individuals express their choices of services characteristics in certain contexts. Therefore, preferences provide a powerful mechanism to influence the behavior of I-centric Services by giving them explicit instructions. Considering the example above again, the individual might have the preference not to be disturbed by incoming phone calls in such situations.

Preferences: Preferences are conditional **choices of service characteristics of an object** depending on context and ambient information. Preferences are applied to objects during the activation of a context.

I-centric Services evaluate preferences to adapt their behavior to what is 'really wanted' by an individual in a certain environment at certain moment in time. Therefore, preferences have to be either gathered from individuals interactively or automated by monitoring, and they have to be expressed in a machine computable form. Gathering preferences interactively causes a lot of interaction between the I-centric system and an individual, who could feel bothered after a short period. A more desirable approach is the collection of preferences based on monitoring the behavior of an individual over time. Inference mechanisms can be applied then to the collected data to predict what preferences a certain individual might have. This process has to run continuously to keep the preferences up-to-date or to precise them step-by-step.

The description of preferences, which can be processed automatically, is another challenging task. Preferences can capture many aspects like mood, interests, live stage etc. that are even hard to describe in words. Furthermore, the kind of preferences that are relevant to different individuals may differ completely. A model for describing preferences must be as generic as possible to avoid restrictions that might prevent the expression of a certain preference. On the other hand, the model has to provide some structuring or categories to allow the assignment of preferences to a certain I-centric Service.

Preferences should be as precise as possible. One approach to teach this is to relate preferences to contexts and to information describing environmental conditions. Continuing the example given in the beginning of this section, only the existence of environmental conditions allows the specification of 'a silent room'. Beside the noise level, many other environmental conditions might be sensed by I-centric communications systems. To emphasis the variety of such kind of information, the term **ambient information** has been coined by the ISTAG group in its 'Scenarios for Ambient Intelligence for 2010' [ISTAG].

In general, ambient information is information that can be collected, gathered, or sensed from the environment. Ambient information comprises temporal and spatial characteristics like any user input, temperature, noise level, light intensity, and presence of other people just to give a few examples. Ambient information is sensed by sensing facilities, like motion detectors or microphones, and transmitted through sensor networks. Ambient information may also include geographical information (e.g. location), environmental information (e.g. temperature), and life conditions (e.g. blood pressure).

Ambient Information: Ambient information is information that can be collected, gathered, or sensed from the physical environment using the objects of the individual communication space of a certain individual.

Furthermore, ambient information is the basis for ambient-awareness, which is introduced in section 3.2.5. A semantic model is needed to describe preferences and ambient information. Such kind of model incorporates knowledge representation to qualify available information and ontology languages to relate syntax and semantic to each other (e.g. Semantic Web [W-SemW], Agent Ontology Service [W-AOS]). The focus for I-centric communications here is to define a

harmonized semantic model that includes human aspects as well as the process to gather, store, evaluate, and exchange preferences as well as ambient information.

2.2.3 I-centric Service

I-centric Services define, manage, and (de)activate contexts in an individual communication space taking the preferences of individuals and ambient information into account. They support an individual (I-centric), adaptive, personalized, and ambient-aware, way to interact with objects in individual communication spaces. Based on the evaluation of 'profiles' that describe preferences, service capabilities, and sensed information about its actual environment, the individual can be provided with personalized services for his actual demands.

I-centric Service: I-centric Services define, manage, and (de)activate contexts in an individual communication space taking ambient information and the preferences of an individual into account. They support an adaptive, personalized, and ambient-aware way to interact with objects in individual communication spaces.

Self-learning capabilities are used to profile the behavior of individuals. Numerous services or several features of different services are combined on-demand. Appropriate terminals and adaptation strategies are evaluated.

I-centric Services need ambient information in order to adapt to the environment. Temporal and spatial characteristics are only two examples of information, which may affect the service behavior. Note, that a certain environment can restrict the functionality requested in a certain context. Interacting in a 'TV context' while driving a car may reduce the available functionality to '*record the movie for later viewing*' or to listen just to the audio part.

I-centric Services activate contexts by choosing the equipment to be controlled (presentation terminals, handhelds, microelectronic controlled devices), their quality of service (volume, brightness, etc.) to be finally connected via heterogeneous networks (BAN, PAN, LAN, WAN) to create an I-virtual private network (see section 2.2.1). The process of choosing and controlling the equipment of the physical environment is supervised by the service logic of I-centric Services. The service logic controls the activation of contexts by combining multiple objects dynamically. It parameterizes objects by defining what, when, and how one or more objects behave in a given condition.

The service logic decides based on profiles and on the status of the objects how those objects should behave in a certain situation. This enables sensitive services that adapt to the environment dynamically. Therefore, mechanisms have to be developed, which enable to gather and to evaluate profiles. The profiles and the status will be evaluated by a domain-specific service logic that also controls the object(s).

Nowadays, service logic is in most cases 'hard-coded'. Once implemented, it cannot be changed afterwards. The basic idea of I-centric communications, to provide individuals with their **own** services that might change over time, requires a more flexible approach. A generic model for describing service logic independent from any execution environment is needed. The description of service logic has to be an input parameter, like ambient information, during the execution of an I-centric Service. That will allow altering the service behavior according to changing situations. Moreover, creating new or altering existing service logic can become an interactive or automated process. Self-learning systems with automatic reasoning engines or interactive tools can be applied then.

The process of creating or modifying I-centric Services has to be accompanied by ontology definitions that describe what services an object is providing. Interactive applications are envisaged that allow individuals to assemble their service by simple 'drag and drop' mechanism. Like a LEGO™ toolbox, the individual should be able to create and to deploy its I-centric Services.

2.3 Reference Model for I-centric Communications

Following the vision explained in section 2.1, this section introduces the reference model for I-centric communications. The work on the reference model has been accompanied by three international activities. They are described in the end of this section.

Figure 8 shows the reference model for I-centric communications. It follows a top-down approach starting with the introduction of **individual communication spaces**, related **contexts,** and **objects**. In general, the topmost layer recalls the I-centric vision that human beings interact with objects of their communication in a certain context.

Furthermore, it is common understanding that I-centric Services have at least to support three different features, namely **ambient-awareness**, **personalization**, and **adaptability**, which will be explained in sections 2.3.2, 2.3.3, and 2.3.4 in detail.

To emphasis that these features are needed for I-centric communications they have been assigned to the individual communication space in the reference model.

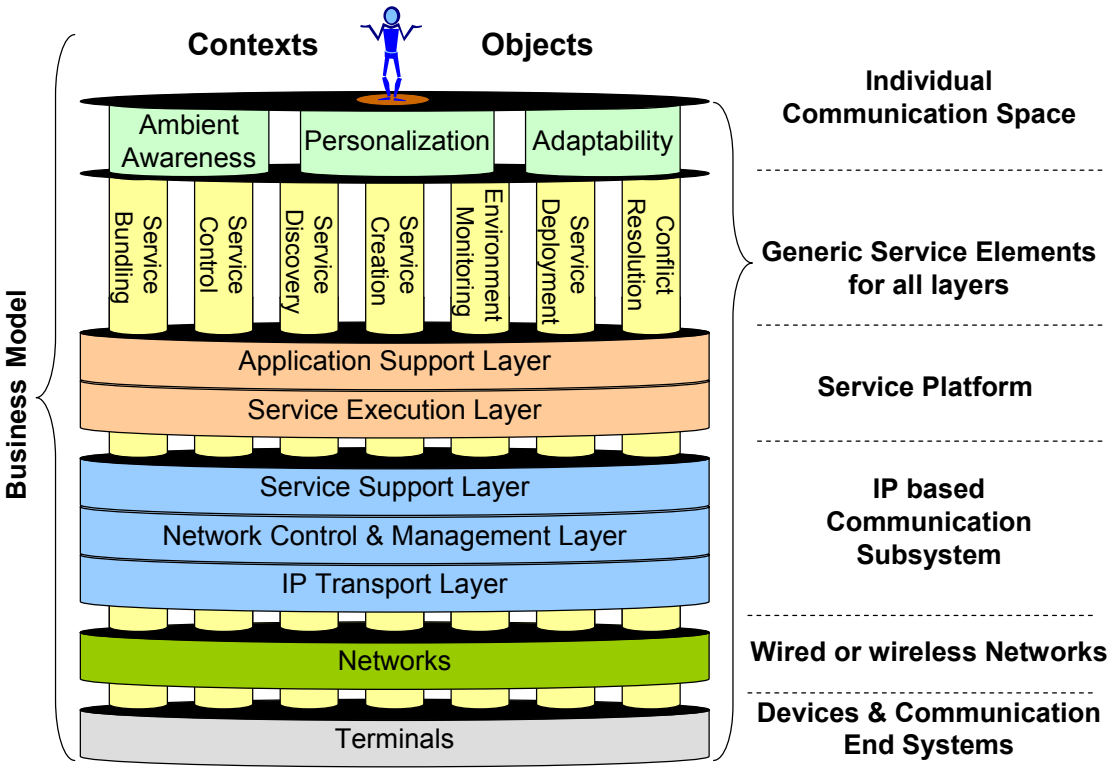


Figure 8: Reference Model for I-centric Communications

The **service platform** for I-centric communications is responsible for shaping the communication system, based on **individual communication spaces**, **contexts**, **preferences**, and **ambient information**.

Preferences will be provided by the personalization feature, whereas ambient information has to be provided by ambient-awareness feature. More details are given in section 2.3.5, which introduces the service platform for I-centric communications.

The **IP based communication subsystem** is responsible for providing the linkage between different objects in the communication spaces. These links have to be maintained and managed even when they are subject to change because of roaming between different network topologies or access networks. There might be non-IP based communication networks underneath the IP-based communication subsystem. They have to be wrapped by bridging facilities (e.g. by

SmartIP devices[15]) to include them in I-centric communications systems. IP communication is seen as the common denominator to harmonize heterogeneous network infrastructures. The IP based communication subsystem consists of three layers:

- Service Support Layer, which provides well defined APIs for the service platform to access the IP based communication subsystem
- Network Control & Management Layer, which combines the traditional concepts of network management with required real-time aspects needed for system wide control functions
- IP Transport Layer, which basically represents OSI layer four

The Wired or wireless Networks layer implements all aspects of the physical connection(s) between different objects. Due to the hierarchical structure of the reference model, a connection in the IP based communication subsystem might use multiple connections in underlying network.

Devices and Communication End Systems provide the physical infrastructure that hosts all other layers. It can be instantiated as a switch responsible for connecting different networks or even as a multi-modal terminal able to interact with a certain individual.

The main features of I-centric communications (ambient-awareness, personalization, and adaptability) affect all layers. Therefore, supporting functions have to be provided as a vertical solution. The reference model introduces the concept of **Generic Service Elements** that implements common functionalities on all layers.

Generic Service Element can be seen as a toolbox from which complex services can be assembled and executed dynamically. The vertical approach allows I-centricity on all layers, e.g. to establish I-centric private virtual networks (section 2.2.3). More details of Generic Service Elements are given in section 2.3.6.

Accompanying to all these technical issues, the **Business Model** for I-centric communication identifies the relationships and information flows between all active roles within an I-centric system. The business model will help to identify reference points between all involved entities and will assist the assessment of the applicability of I-centric Services for different business domains.

The IP based communication subsystem, the wired/wireless networks, and all terminal aspects are out of the scope of this thesis. The work is based on the assumption that the APIs provided by the Service Support Layer will cover all necessary functionalities to perform the tasks of I-centric communications. Therefore, the service platform defines a set of requirements to these APIs (for more details see section 2.3.5).

In the following, the building blocks of the reference model for I-centric communications will be discussed in more detail.

2.3.1 Business Model

The vision of I-centric communications requires new business models. The borders between traditional roles and administrative domains: network provider, content provider, service provider, and retailer are blurring.

An individual may become service provider (ad-hoc and peer-to-peer), or content provider, or retailer. Additionally, roles may change dependant on context, which implies a very flexible business model.

---

[15] SmartIP devices introduce new kinds of computing facilities into the IP world. Nearly each kind of device can be equipped with IP technology enabling the global accessibility of these devices. Even large groups of devices using proprietary networking technologies can be wrapped by a single SmartIP device providing address translation to reach each of the non-IP devices.

The business model for I-centric communication has to cover:

- roles, relationships, and reference points
- business topologies
- service lifecycle (creation, deployment, management, and billing)
- benefits for parties involved in the market value network

Business Model: A business model is a description of how an entity or a set of entities intend to create value with a product or service. It defines the product or service, the roles and relations of the entity, its customers, partners and suppliers, and the physical, virtual, and financial flows between them. [W-WG2-BM]

The first objective of a business model for I-centric communication is a model for describing relationships between involved parties in a global business community. Based on these relationships, roles and reference points will be defined. This allows the participation of each business partner on a global business on one side and provides the freedom in development and integration on the other side. Reference points provide standardized points of contact and information exchange between business partners. Such a business model for I-centric communications is a prerequisite for the definition of a service architecture that supports the required functionalities of the whole business life cycle.

2.3.2 Personalization

Personalization is considered as being the key factor for I-centric communications. Information and services must become increasingly tailored to individual preferences to make the usage of services easier and the perception of the individual communication space richer.

An extended personalization concept is needed that enables value networks (e.g. value chains) of content providers, network providers, and service providers to offer personalized services to mobile individuals in a way that suits their needs, at a specific place and time.

Personalization aspects like preferences, role, and task, have to be integrated and the relation between the aspects must be studied. Investigations into benefits according to the user perception are necessary and needs to be considered in the design of personalized services and personalization supporting frameworks.

For I-centric communications, this means that objects available in an individual communications space have to adapt to the preferences of individuals. Personalization models each individual in the I-centric Service platform by managing its preferences and providing these preferences to I-centric Services.

Personalization: Personalization provides the information for modeling preferences for an individual communication space in the I-centric system. [W-WG2-Per]

To reach this goal, personalization federates profile information (containing preferences). Personalization incorporates dynamic behavior to enrich stored and federated information to enable pro-active I-centric Services. This leads to an overall profiling infrastructure managing the individual preferences.

The main research issues behind personalization are:

- how to gain personal preferences from individuals (interactively or automated)
- how to store these preferences in profiles (profile format & categories),
- standards to exchange profiles,
- secure privacy sensitive parts of profiles, and
- how to describe active contexts using individual preferences.

2.3.3 Ambient-Awareness

In I-centric systems, services will be tailored to contexts of the individual communication spaces. The services will automatically adapt themselves to changes in the environment. The services have to deal with a dynamic environment of nomadic individuals. The adaptation to the situation (in a certain context) is hidden from the individual. It is provided with optimal experiences and benefit. Vice versa, the environment can be influenced by the presence and activities of the individual and adapt itself accordingly.

Therefore, ambient-awareness deals with gathering ambient information from network, application, individual, terminal, and contexts. The federation of ambient information from various sources, according to individual's mobility and roaming is an integral part of ambient-awareness. Intelligent inference systems for missing information are needed in order to incorporate as much information as possible to provide an automatic, ambient-aware environment to the individual.

Ambient-awareness: Ambient-awareness is the functionality provided by an I-centric system to sense and exchange information about the current environment, an individual is in at a certain moment in time. [W-WG2-AA]

Sensors networks will play a major role in providing ambient information. Sensor technologies will be embedded in mobile equipment, communication networks, living and working environments to sense who the user is, where he is, what he is doing, what the environmental conditions are, to provide this ambient information to I-centric Services.

Advances in sensor technology are conditions to reach further adaptation of services to the environment of individuals. Many devices in our current world already adapt in some form to their operating environment. One example is the television set that adjusts its image contrast to the ambient lighting level. 'Intelligent I/O behavior' can be used to adapt the output characteristics depending on the context, but also the input characteristics should change in many situations (do not bother the individual with long lasting interactions, but combine the context information with the basic intention that was beforehand communicated by a few interactions). This intelligent I/O behavior demands the development of multi-mode user interfaces and corresponding support functions in the various consumer devices as well as in the service adaptation network.

2.3.4 Adaptability

Section 2.3.2 (Personalization) and section 2.3.3 (Ambient-awareness) have introduced two main functionalities of I-centric Services. Adaptability is the third one and is mainly based on information provided by personalization and ambient-awareness. It provides the functionality to adapt I-centric Services to personal preferences and environmental conditions. Therefore, adaptability can be seen as a function that activates a context based on whatever information is provided by ambient-awareness and personalization.

In general, I-centric adaptability translates the wishes of individuals, which are usually inaccurate, incomplete and sometimes even contradictory, into a set of rules precise enough for processing to be automated with sufficient reliability. It has implications in the structure of the services to allow adaptability and is the engine, which activates a context at a certain moment in time in a certain environment. [W-WG2-Ad]

Adaptability: Adaptability is the functionality provided by an I-centric Service to activate context taking ambient information and individual preferences into account. An active context might be adapted continuously to reflect changes of individual preferences or environmental conditions over time. [W-WG2-Ad]

Typical situations when adaptation takes place include a substantial change in characteristics of connectivity, entering into a new service domain, or changing terminal device in service session.

By technical means, adaptability requires the adaptation of media, content, and service behavior. During the last years, a variety of concepts for adaptation has been developed [Arb00b, Arb00d, Arb00g, Pfe99]:

- communication streams can be altered during transmission (e.g. bit rate adaptation),
- media types can be changed (e.g. text-to-speech conversion),
- type of presentation can be adapted (e.g. downscaling an image to fit a PDA screen),
- altering the content of a message (e.g. adding or stripping off information), or
- modifying the service behavior (e.g. by customer service control functions).

Adaptability cannot be only reactive. When the battery of a mobile device dies or the connectivity breaks, many actions become impossible. However, something could have been done beforehand. Therefore, adaptation must also be proactive, which in turn requires predictability of the near future. An important question in predictions is to distinguish between the situations in which the human behavior seems to be predictable and those being unpredictable.

2.3.5 Service Platform for I-centric Communications

A Service Platform for I-centric communications is responsible for shaping the communication system, based on individual communication spaces, contexts, preferences, and ambient information. Finally, it (de)activates objects (advised by I-centric Service), identifies causalities between them based on sensed environmental data, controls the services offered by these objects, and converts data structures and operations for interworking between services. The equipment is configured dynamically, its state is profiled, distributed objects are controlled, service creation and deployment are supervised, and the interworking among domains is enabled by the platform.

Service Platform: A service platform is an infrastructure that supports the development and operation of I-centric Services by providing a set of service features:

- execution environment for services and objects
- supports (re-)deployment (hot-plugging) of services and objects
- supports the (re-)binding (configuration) of services and objects
- supports the interworking of services and objects

The service platform consists of two layers:

- The Application Support Layer provides well-defined APIs to applications, services, and objects. It offers generic service elements that can be used by developers of these entities to ease and fasten the process of design, implementation, deployment, and management.
- Service Execution Layer provides the actual runtime environment of applications, services, and objects. It supports their secure, QoS aware and managed execution.

Moreover, the service platform provides functional blocks that directly support the I-centric approach. These functional blocks manage ambient information, preferences, and adaptability to be offered to I-centric Services. To fulfill the functionalities requested by I-centric communications, I-centric Service platforms have requirements on the underlying communication subsystem.

This is caused mainly by the empowerment of any individual to act as a service provider or network provider in a paradigm shift from a provider centric paradigm to a decentralized I-centric paradigm. From an I-centric perspective, this is done by sharing objects between different individuals or by allowing another individual to use objects out of ‘my’ individual communication space.

On the other hand, the requirements are based on information that has to be provided by lower layers to the service platform. Traditional platform approaches (e.g. object-oriented middleware platforms) try to hide as much as possible technical parameter between the different layers. I-centric Services have to be provided with ambient information. Therefore, the traditional con-

cept of transparencies [ODP, X.901-904] vanishes. Ambient information has to be provided trough all layers. Only the I-centric Service itself can decide what information is useful or not. Intelligent filtering mechanisms and translation rules will help the I-centric Services to 'understand' the ambient information.

By technical means the requirements of the service platform to the underlying network are:

- peer-to-peer, ad-hoc communication (single or multi-hop, ad-hoc relays)
- bearer technology aware stacks and applications
- QoS in various areas (guaranteed streaming, data-rate, real time traffic)
- global roaming by integrating access networks supporting various wireless and wire-line bearers, various radio interfaces, and mobile-IP
- ubiquitous Addressing (Address mapping, Routing over Address-less networks)
- multicast services (incl. advanced addressing methods like geo-cast)
- Personal Identifiers to overcome address schemes of traditional communication networks
- sensor networks for gathering ambient information
- privacy, assurance of the confidentiality of data
- integrity, assurance that the content of a transmission cannot be altered during transmission
- confidentiality, assurance of the confidentiality of information
- non-repudiation, assurance that the sender (receiver) cannot deny having sent (received) information
- authentication, assurance of the identity of the sender and the receiver of information
- access control/authorization, Assurance that only authorized people can access information
- accounting/billing. Transparent billing for service usage and integrated network access

However, the paradigm shift addressed here does not only concern the individual as a provider of network related services. A service platform allowing global mobility and transparent access to any kind of service over a common IP platform is the basis for allowing everyone to provide a wide range of services [W-WG2-BM].

2.3.6 Generic Service Elements

I-centric communications systems will have to cope with issues like numerous service providers, always-connected individuals, automatic service adaptation, and ambient-awareness. Aspects like dynamic service discovery and service provisioning in (for individuals and services) unknown environments and personalized services usage requires new mechanisms to support I-centric communications systems.

To simplify the definition and realization of I-centric Services and applications, a set of reusable software components will support functionalities common for different services and applications. These components are called Generic Service Elements to emphasis their general applicability for all kind of services.

Generic Service Elements: A GSE is a functional software component that can be used by other GSEs, services, or applications and it is hosted by the I-centric Service platform. GSEs provide functionalities common to different services and applications to ease and shorten their development process.

Because I-centric Services should work under changing environmental conditions, serving changing individual preferences, the most promising candidates for common functions are:

- Service Discovery (a mechanism to discover service features dynamically that are provided within a certain environment or by a certain physical resource)
- Service Management (how to manage context (dynamic relationships and causalities between individuals and their environment))
- Service Deployment (how to deploy services in distributed environments)
- Service Composition (dynamic interworking of services will help to create and operate contexts)

- Service Logic (the evaluation of the preferences and ambient information leading to a decision what has to be done by an I-centric Services)
- Service Control (the process to control all the resources needed for a specific purpose)
- Environment Monitoring (how to gain ambient information)
- Reservation (to manage exclusive usage of objects)

By technical means, objects and generic service elements are quite similar. They have to provide well-defined interfaces to be used for service developments. Generic service elements can also be seen as enabler for objects by providing functionalities that are common for all objects.

Consequently, well-defined collections of interface specifications designed for certain business domains are needed. The idea is to equip same kinds of objects with standardized interfaces for functional (usage) and non-functional (management) interfaces. Especially, from the area of telecommunication, Open Service APIs like OSA/Parlay [W-OSA] build the basis for such interfaces.

As the vision for future mobile systems does not cover communication scenarios only, new Open Service APIs have to be developed. On the one hand, the service infrastructure should be opened by means of Open APIs to enable easy of use service creation.

Such APIs must provide framework functionality, like hot plugging of services, dynamic (re-)binding (e.g. in ad-hoc environments), service mobility, AAA-services, support for automated SLA negotiation [FAIN], contracting, and so on. It is to remark, that this generative approach may require well-defined internal API's and callbacks, too.

2.4 State of the Art

In the following, four activities are briefly introduced, which have influenced the work on the reference model for I-centric communications. Namely, they are the Unified Messaging Research project, the IST projects WSI and WWRI, as well as the activities ongoing in WWRF's Working Group 2.

2.4.1 Unified Messaging Research Project

The concept of Unified Messaging has emerged from the research of Personal Communication Support [Eck96]. Reaching industrial relevance it addresses the task of overcoming the multiple-mailbox approach of today's communication scenarios with separated facilities for e-mail, voice storage, fax reception, etc. This coincides with the vision of I-centric communications, to deliver information any time, any place, in any form, as it is also described in the concept of the UMTS Virtual Home Environment [UMTS]

A unified messaging system has been implemented within the Unified Messaging research project[16] [Arb99a]. The project has addressed service access and service delivery on both, fixed and mobile terminals. To support this variety of end systems, a special capability of the developed messaging system is the selection of the most appropriate terminal or application for an incoming message. The most appropriate terminal will be determined by means of its availability, status, and capability to handle a certain communication service. If no appropriate terminal can be found, a pool of converters adapts the communication media to the available terminal on demand.

The provision of Customer Service Control Capabilities, enabling users to define rules for the handling of incoming calls and messages according to their individual preferences, is another important feature of the system. The messaging system acts as an electronic secretary. Once advised by the user, it is responsible for controlling the user's communication environment, i.e. telephone calls or e-mails. Each service is processed based on user-predefined rules.

[16] The project was a research collaboration between the Fraunhofer Institute FOKUS and the Department for Open Communication Systems at the Technical University of Berlin.

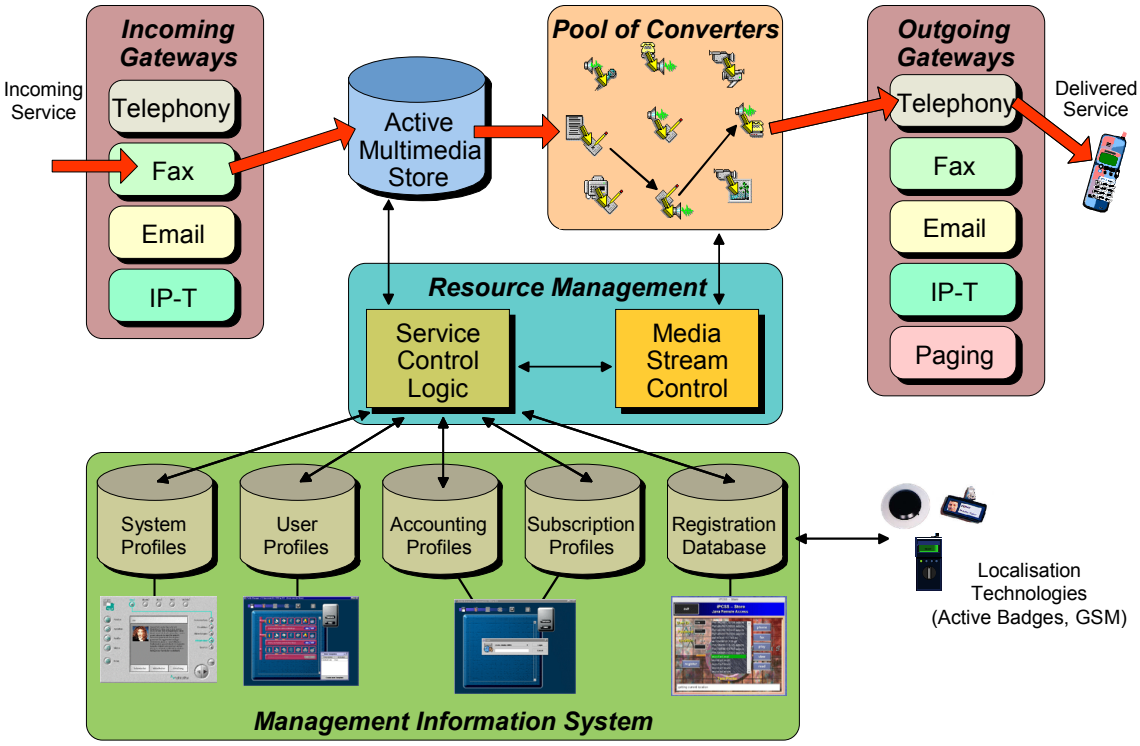


Figure 9: OKS' Unified Messaging Systems Functional Components

The messaging system incorporates the following features:

- Call Screening, Call Scheduling, Call Forwarding
- Customer Service Control
- Adaptation of Services and Automated Content Adaptation
- Multimedia Message Storage and Remote Message Access [Pfe98]
- Synchronous and Asynchronous Service Delivery
- User Registration at Terminals and Locations (automated, manually, or scheduled)

To be able to adapt easily to new bearer networks, services, and applications, the messaging system separates service access, service logic, and service control. Specific parts of the system should be responsible for the access to particular bearer networks, while generic parts take care of the service logic and the service processing (the establishment of bearer connections including necessary service conversions).

A *Customer Service Control Component* allows users to configure their individual preferences using a rule based *Service Control Matrix*. In rules a user can define when, where, for whom, and using which medium or terminal he wants to receive a message. A setting likes 'In the next two weeks, my mobile phone with the number 123456 should only ring if my girlfriend wants to reach me', could be described in two rules:

- Between day X and Day Y, if user Z calls me → forward to my telephone 123456.
- Between day X and Day Y, if calling user is not Z → forward to my voice-mail-box.

A user can have several rules. The rules of a user will be evaluated for each incoming communication request. The results of an evaluation directly affect the *Service Control Component*, which handles the communication request as the user has configured it.

The realized Unified Messaging System implemented some functions that have been taken into account during the specification of the architectural framework for I-centric communications (e.g. Customer Service Control and selection of appropriate terminals, for certain communication requests).

2.4.2 IST Project Wireless Strategic Initiative

The reference model developed by the Wireless Strategic Initiative (WSI) [W-WSI, Arb03b] describes the Wireless World as a set of concentric spheres inhabited by networked Communication Elements (CEs). CEs are the generic representation of devices and nodes in the Wireless World. The functions incorporated in a CE are provided by different building blocks.

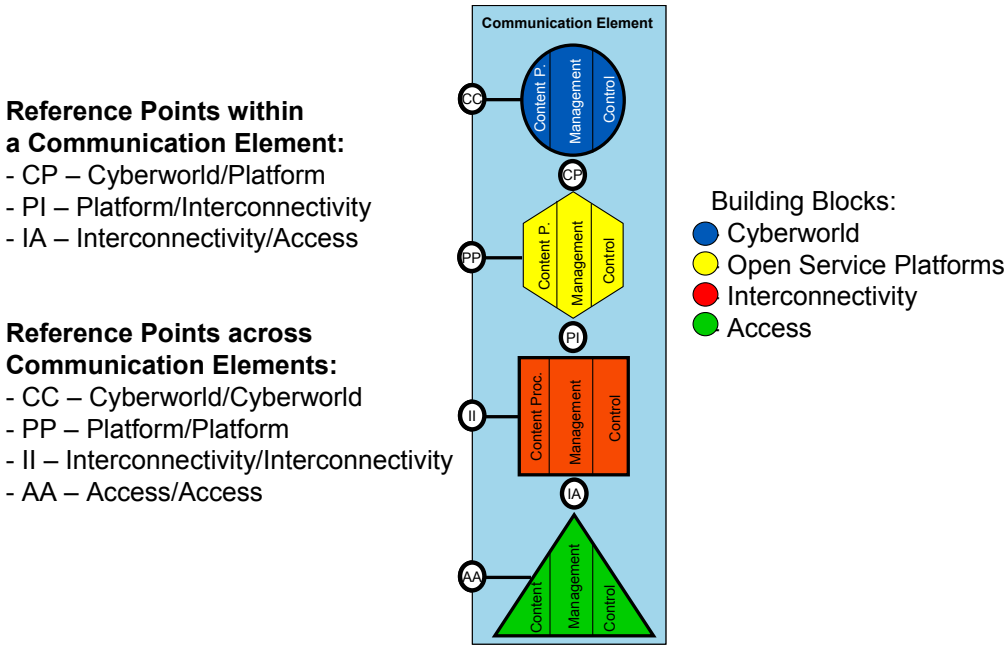


Figure 10: WSI Reference Model

The identified building blocks are namely:

- The **Cyberworld** block, hosts all application-specific functions and manages their characteristics to ensure the underlying infrastructure is being used efficiently to satisfy the user.
- The **Open Service Platform** block, is responsible for providing a service infrastructure to facilitate the creation of services according to user and operator needs.
- The **Interconnectivity** block is the Networking part of the Wireless World reference model. The functions located there take care of linking together resources.
- The **Access** block implements all aspects of physical connections between different entities.

Reference points have been identified between the building blocks of a Communication Element and among different Communication Elements. The reference model for I-centric communications has been developed in parallel to the WSI project. Therefore, the two reference models are very similar. The I-centric model is focusing more on the user perspective, whereas the WSI model was more oriented on the networking and radio aspects.

2.4.3 IST Project Wireless World Research Initiative

The strategic objective of WWRI project [W-WWRI] was to provide a launch pad to the wireless community (industry and academics) for the development of a balanced cooperative research program addressing the future Wireless World.

The concrete tasks have been to produce a report on the wireless industry sector and its likely developments in order to be able to establish the strategic research directions for the wireless sector in the timeframe until 2010, and to establish and propose technical areas of research (Figure 11) as well as an evolution roadmap for systems 3G beyond.

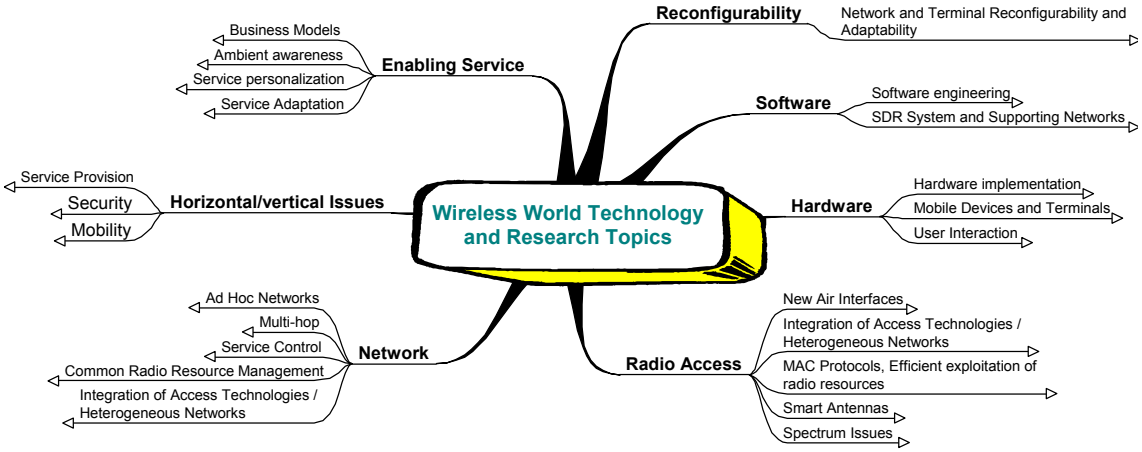


Figure 11: Overview on technology and research areas structuring

The results have mainly influenced the orientation of I-centric communication towards ambient-awareness, personalization, and adaptability. These issues have been identified as mandatory for future services based on a broad consensus between mobile operators, manufacturer, and academics.

2.4.4 Wireless World Research Forum

The Wireless World Research Forum (WWRF) [W-WWRF, Mohr03] has the objective of formulating visions on long-term strategic research directions in the wireless field, involving industry and academia. The aim is to generate, identify, and promote research areas and technical trends for mobile and wireless system technologies. In the 'Book of Visions 2001' [W-BoV01], the forum has identified needed research items towards wireless systems 3G and beyond.

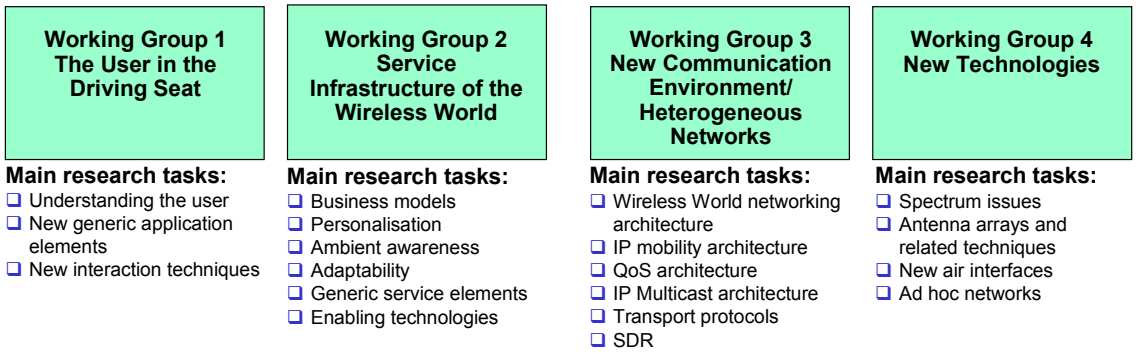


Figure 12: WWRF – Working Group Structure

The vision of I-centric communication has been adopted by the WWRF. It builds the basis for the definition of the service architecture for the Wireless World. Especially WWRF working group 2 is investigating service architectures that support I-centric applications.

2.5 Aims of this Thesis

A Service Architecture compliant to the Reference Model for I-centric communications is needed, to implement an I-centric communications system. The Service Architecture has to define building blocks and their interworking to provide the functionality requested by the reference model. The technical areas of concern have to be identified to define the necessary system components.

The Reference Model for I-centric communications requests for list of components and features that are briefly recalled adding the areas of concern to be analyzed.

Mechanisms to control telecommunication devices, home / office appliances, and in-car electronics in a unified way are needed to model and realize the objects of individual communication spaces.

The service scenarios, expected to become possible with I-centric communications, require the design of interaction mechanisms for distributed objects. Furthermore, object and service discovery mechanisms, and object repositories are needed to enable intelligent service brokerage. Service and object ontologies facilitate these discovery and brokerage mechanisms.

Profiling, decision making, and intelligent device control has to be embedded in the business logic of I-centric Services. A user/context model for I-centric communications is needed to feed the business logic of personalized, ambient-aware, and adaptive services.

Service Composition and Service Bundling are needed to combine the services provided by the objects of individual communication spaces.

Due to the variety of objects and services to be combined in I-centric Service, a Service creation environment is needed that suits the requirements of individual users. The graphical user interface of such a creation environment is not the focus of this work but the platform support to enable such kind of easy-to-use service creation tools.

To come up with a coherent service architecture for I-centric communications systems all these aspects have to be integrated into a single framework. The next chapter introduces the architectural framework for I-centric communications, which has been derived from the reference model described in section 2.3. The Open Profiling Framework provides the concepts to realize ambient-aware, personalized, and adaptive services, whereas Super Distributed Objects provide the infrastructure, on which I-centric Service can be realized.

3 Architectural Framework

This chapter introduces the architectural framework for I-centric communications systems. The main components and their interactions are described to set the ground for ambient-aware, personalized, and adaptive services. A special emphasis is given to a model to describe and handle information that is needed for I-centric communications. Based on this model, the architectural framework supports advanced functions for user initiated services creation and distributed services execution on top of heterogeneous network environments.

3.1 Introduction

The Reference Model for I-centric communications (section 2.3) specifies the main building blocks for I-centric communications systems. To realize a communication system following this reference model, the building blocks have been assigned to two different research activities. The first one aims for the provisioning of ambient-aware, personalized, and adaptive services (**Open Profiling Framework**), whereas the second one concentrates on modeling and controlling the objects of the individual communication space and their physical resources underneath (**Super Distributed Objects**).

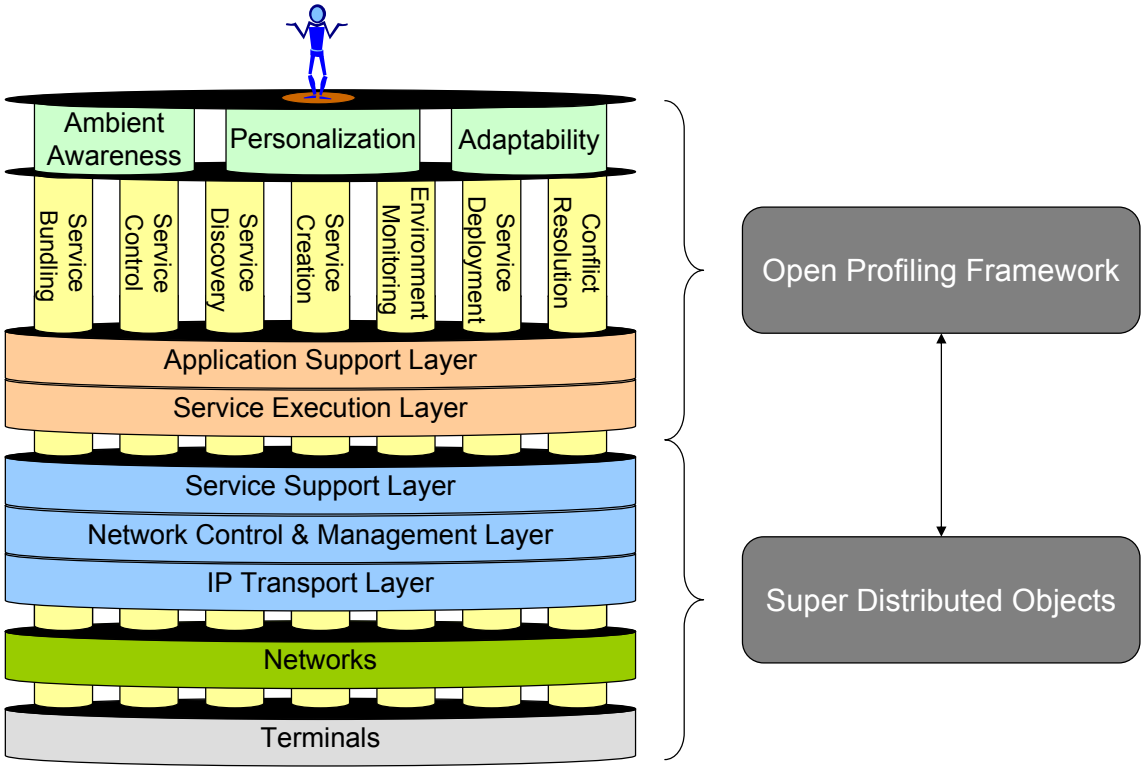


Figure 13: Reference Model vs. Architectural Framework

The **Open Profiling Framework** picks up the idea of providing a service architecture for I-centric communications (section 2.3.5) that inherently support ambient-awareness (section 2.3.3), personalization (section 2.3.2), and adaptability (section 2.3.4). A general support for these features must be embedded in the service architecture, because future telecommunication services are expected to provide them [Arb01d, Arb03a]. This will enable reusable software components that will be part of the service architecture ready to be used by several applications and services. To come up with a coherent model, where ambient-awareness, personalization, and adaptability are really incorporated, they have been integrated in the Open Profiling Framework. This framework represents the upper part of the reference model for I-centric communications as depicted in Figure 13. The research undertaken for the Open Profiling Framework concentrates on the definition, creation, and execution of I-centric Services. The Open

Profiling Framework, as introduced in section 3.2, abstracts from underlying resources by defining a generic **usage interface** for I-centric objects and according ontology definitions for them. Ontology definitions are used for the process of service creation, service bundling, and for service discovery.

Super Distributed Objects represent the objects of individual communication spaces. They implement the physical execution of service requests and provide their functionality via well-defined interfaces compliant to the Open Profiling Framework (usage interface). [Arb01f] The research in this area is mainly focused on unified models for the abstraction from heterogeneous and distributed computing resources.

Beside the execution of service request, SDOs have to support a number of so-called non-functional aspects, which are mainly related to management issues (e.g. monitoring, or configuration). These functions are needed for the runtime environment of Super Distributed Objects enabling the setup of large-scale distributed systems, which are needed to realize individual communication spaces.

3.2 Open Profiling Framework

The general idea behind the Open Profiling Framework has been derived from the Reference Model for I-centric communications. I-centric Services have to activate contexts according to user preferences and ambient information to adapt/control the I-centric telecommunication system (Figure 14). Based on this information, I-centric Services decide what to do under certain circumstances (more precise: the service logic inside I-centric Service takes this decision).

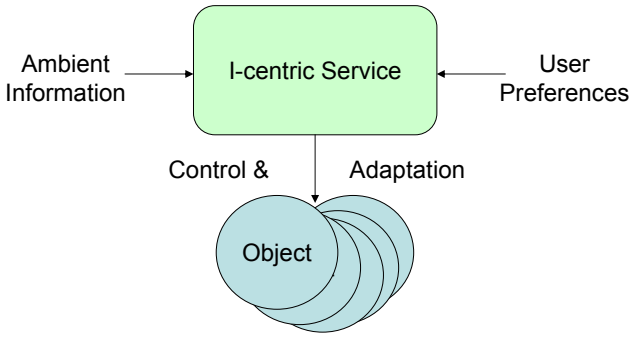


Figure 14: Open Profiling - General Idea

To support the service logic with harmonized and well-defined information, the Open Profiling Framework takes responsibility to manage ambient information and personal preferences. It defines a general syntax and profile structure for defining and storing information. Furthermore, it provides a so-called ontology to combine the syntax with domain-specific semantic.

The aim of the ontology definitions, developed within the Open Profiling Framework is two-fold. On one hand, it is to help individual users to understand the functionality of certain objects, available in their communication space. On the other hand, it enables a matching algorithm that checks whether two or more objects can cooperate by supporting the same ontology.

The management (storage, query, categorizing, and federation) of profiles is another task of the Open Profiling Framework. The profiles contain the information about objects, contexts, personal preferences, and ambient information available in an individual communication space. The interfaces to control objects are also part of the Open Profiling Framework. They have to be aligned with information managed in the profiles. [Arb00f, Arb01d, Arb01f]

3.2.1 Overview

The process of ongoing developments and day-by-day introduction of new terminals and services makes it impossible to define a static set of profiles for preferences, objects, services, or other entities that may characterize a communication space in the future. Therefore, a profiling

framework, which is able to cope with dynamic profiles, has been developed. The framework provides a set of functional components to receive, categorize, store, and evaluate profile information.

3.2.1.1 Overall Structure of the Open Profiling Framework

The framework is responsible for managing objects involved in a context, identifies causalities between them based on sensed ambient information. It controls the services offered by these objects, and converts data structures and operations for the interworking between different objects and services. The main functional blocks of the Open Profiling Framework, shown in Figure 15, are described in the following.

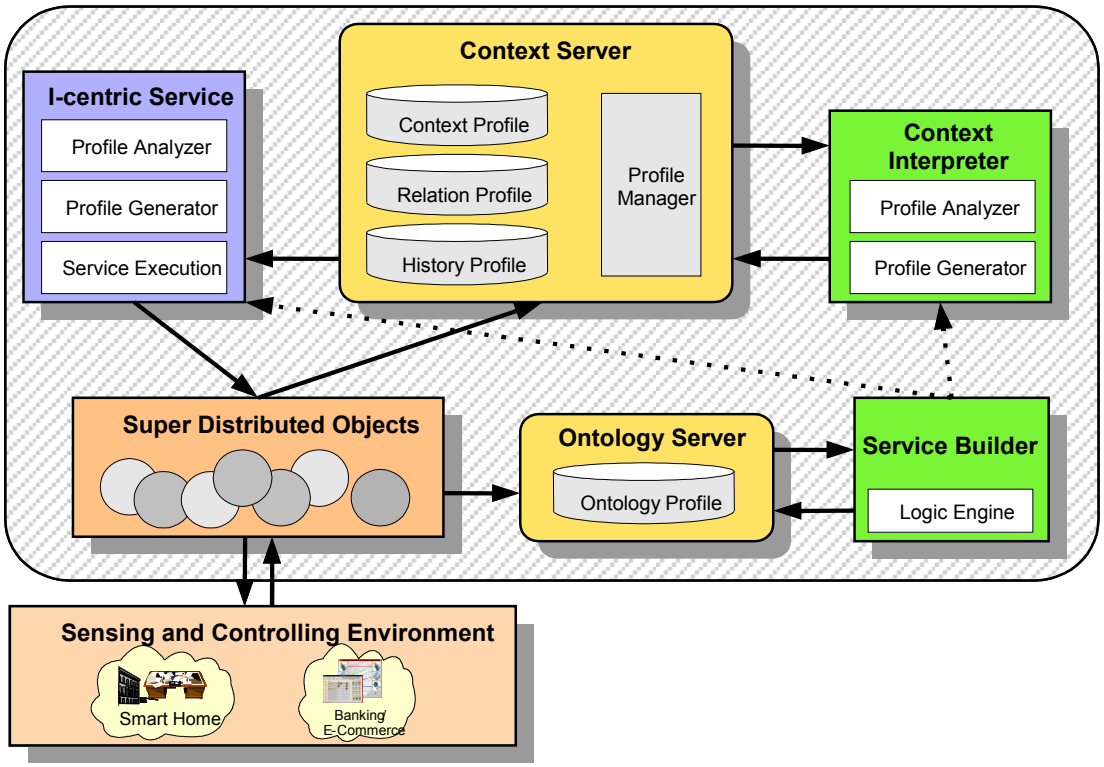


Figure 15: Open Profiling Framework

The Open Profiling Framework (OPF) consists of seven different building blocks:

Sensing and Controlling Environment represents the physical environment augmented by various kinds of sensor technologies, heterogonous network infrastructures, and terminals that provide ambient information for I-centric Services.

Super Distributed Objects (SDOs) hide the heterogeneous character of the underlying environment by providing well-defined interfaces towards the other components of the OPF (SDO usage interface). SDOs are also responsible for converting the proprietary formats of information, which might be used within the Sensing and Controlling Environment into well-defined formats, the other components of the open profiling framework can interpret. Super Distributed Objects gather contextual information that occurs in the physical sensing and controlling environment (e.g. detecting temperature, or any user input) and provide them to the Context Server. Before sending any occurring information to the Context Server, an SDO has to register its ontology (provided functions and data with corresponding human readable description of semantic) with the Ontology Server.

The **Context Server** (CS) collects, categorizes, and stores all ambient information that is send by SDOs. In addition, the Context Server manages the profiles containing personal preferences and relations between objects. It manages three persistent profiles: Context, Relation Profile,

and History Profile. The CS provides intelligent query mechanism for I-centric Services and Context Interpreters to support complex search procedures within individual communication spaces. The Context Server categorizes incoming data by verifying the ontology of the sending SDO. Any incoming data will be stored in a Context Profile. A categorization is assigned to every data record in the Profiling Framework. The categorization offers querying and addressing of profile information. Access to data that is stored in the Context Server is realized through categorizations. A profile entry can be part of different contexts at the same time, because an individual might act in different contexts in parallel (see section 2.2.1). Relationships between different profiles are stored in the *relations profile*. E.g., a relation connects the record of an individual user with the record of a certain location, which can indicate different meanings: 'the user is currently located in this room' or 'this room is the user's office'. The actual meaning is defined by the ontology assigned to each relation.

The **Context Interpreter** is responsible to enrich or simplify Context Server Data by domain specific knowledge and individual preferences[17]. This process enables the definition of abstract services like 'I would like to have it warm when I'm at home'. A Context Interpreter subscribes to certain categories of the Context Server and is notified if corresponding information occur. Ambient information is mapped to higher-level representations, feasible for I-centric Services (new profiles or relations). Occurring information is analyzed and transformed. The transformation determines the usage parameters to control certain SDOs. The result is send back to the Context Server, where it will be stored in the Context Profile.

An **I-centric Service** is responsible for the (de)activation of contexts by taking all information, available in the CS, into account. Each I-centric Service is dedicated to a single individual, serving special wishes, defined within its service logic (profile evaluation mechanisms). An I-centric Service can subscribe to all changes of location information of a certain individual. This mechanism enables I-centric Services to react immediately if anything is changing in corresponding profiles. The I-centric Service resolves 'what has to be done by SDOs' - using its inherent service logic - and instructs corresponding SDOs accordingly. The algorithm, how a context has to be evaluated, is specified in a generic way. Each kind of Context Interpreter or I-centric Service is parameterized with an algorithm that is designed for a special purpose of that I-centric Service. This follows the vision of the unified model for service personalization, where each service will be processed in a different way facing only the demands of a certain individual.

Ontology Server manages ontology definitions for all information that is handled within the Open Profiling Framework. It is not possible to handle information inside the Open Profiling Framework, which has not been assigned to certain ontology. That prevents the framework from being corrupted by non-valid information that cannot be processed. The **Ontology Server** provides functions that can be used either by the SDO to insert, modify, or remove its ontology or by the Context Server to validate, compare, or translate ontologies. SDOs are pre-configured with certain ontology. If an SDO registers with the Ontology Server, its ontology becomes known to the Open Profiling Framework. The Ontology Server manages all ontologies that are valid within the framework. For instance, the registration process of an SDO will lead to a unique ontology ID that is used by other components to refer to that very ontology. Ontology defines the meaning and syntax of information that can be generated or consumed by an SDO (e.g. the command 'light on' will be consumed by an SDO that represents a certain light). Each component of the framework can check automatically, whether it acts conform to a certain ontology. The Ontology Server follows the ontology-concept of FIPA organization [W-AOS]. It provides validation and mapping of different ontologies to make services more flexible. For instance, a service that has been designed to control some kind of light (with a certain ontology)

[17] E.g. a temperature sensor sends the information '25 °C' to the Context Server. The Context Server contains relations that assign this very sensor to a certain room (i.e. bathroom), and individual preferences that define 25 °C as 'warm' for a certain user. A Context Interpreter can augment such sets of information to 'it is warm in the bathroom'. The opposite direction is also possible (to map from 'warm' to '25 °C').

can also control other lights (with different ontology) if a mapping between both ontologies is provided.

The **Service Builder** is an interactive application that provides user initiated service creation. Based on ontology definitions, information about concrete SDO instances and personal preferences, the Service Builder generates, deploys, and manages I-centric Services. The Service Builder provides a graphical user interface where condition and actions for certain contexts can be easily defined by an individual user. That means, a user can create and deploy its own services by selecting some conditions (e.g. if I enter a room, or if somebody is calling me) and some desired actions (e.g. switch on the light, or connect call to my current location). After this visual configuration of the service, the Service Builder generates the corresponding I-centric Service and deploys it to the I-centric communications system. No active involvement of a system operator is needed for this kind of user initiated service creation.

3.2.1.2 General Processing of I-centric Services

The Open Profiling Framework provides a general methodology how I-centric Services are processed. That general methodology reflects the integration of ambient-awareness, personalization, and adaptability. I-centric Services react on changing environmental conditions or direct instructions from individuals. As shown in Figure 14, this section will introduce the general processing by explaining one example service.

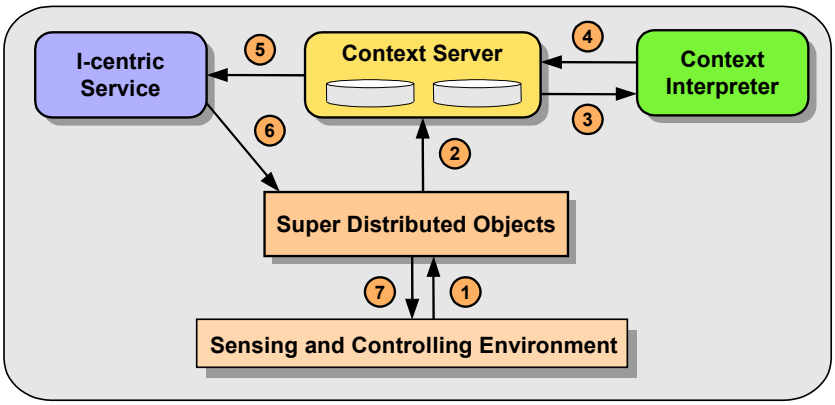


Figure 16: The Open Profiling Framework

E.g., it is assumed, that an individual enters a room in his home and tells the I-centric communications system, that it is too cold. The sentence is recorded by a microphone, which is part of the Sensing and Controlling Environment. The information, detected[18], is send to the respective Super Distributed Object that represents the virtual counterpart of the microphone device (step 1). The SDO maps the information originated in the Service and Controlling Environment to a representation the Context Server, I-centric Service, and Context Interpreter can process. This mapping process generates a representation compliant to the ontology of an SDO. The mapped information about the recorded sentence is send to the Context Server (step 2), where it is stored in the Context Profile. Again, the SDO gathers **ambient information** and sends it to the Context Server.

A Context Interpreter, if it has subscribed to changes of that category, is notified about the changed data (step 3). The Context Interpreter processes the data and generates information

[18] It is assumed here that the Sensing and Controlling Environment converts the spoken words into text, which can be further processed by the SDO. Appropriate Text-to-Speech technologies are available today [Pfe99, Wechs99, W-Lang].

feasible for I-centric Service components[19]. In this case, it could mean that the temperature is too low in a certain room, or that an individual requested to turn on the heater in the room. The generated data is sent back to the Context Server (step 4). If necessary, this step can be repeated involving several instances of Context Interpreters.

An I-centric Service, responsible for this certain individual, has subscribed to changes of the category, the Context Interpreter has provided. If changes happen, the I-centric Service receives the new data (step 5). It processes the information that the temperature is too low in this certain room and possibly comes to the decision that the heater device standing in that room has to be turned on. The **preferences** of the individual user are taken into account to determine the appropriate temperature level. Therefore, it identifies the SDO, which represents a heater device nearby the user and sends a request to the heater's usage interface (step 6). With this request, the I-centric Service demands the activation of the corresponding device. Finally, the SDO communicates with the underlying Sensing and Controlling Environment. It accesses the physical heater device and turns it on (step 7). The activation represents the **adaptation** of the physical environment regarding a certain context. As mentioned before, ambient information, preferences, and adaptability are integrated to fulfill the requirements of I-centric communications.

3.2.2 Business Model

The vision for I-centric communications has a number of implications for future business models [Arb02a]:

- changing constellation of functions and roles
- increased flexibility of roles and actors
- influence of feasible business models on functional architecture

The *increased flexibility of roles and actors* refers to the increasing unbundling of functions and to their peer-to-peer characteristics. The introduced vision implies the convergence of traditional telecommunication systems, Internet based systems, and new applications. The borders between traditional roles (such as network provider, content provider, service provider retailer) and according administrative domains are blurring.

An individual can become service provider (ad-hoc networking, peer-to-peer), or content provider (e.g. document sharing), or retailer. Additionally, the roles may change in the same active context requiring a very flexible business model. Different scenarios on the combination of roles and functions, which in turn influence and are influenced by the business topologies can be envisaged here. The business topology determines where intelligence (service / business logic) is realized (on the terminal or inside the network) and whether the service or the individual user is mobile.

The following figure shows a general business model which has been introduced by TINA-C [W-TINA-BM]. This model is general enough to be applied to any business domain or application field such as I-centric communications systems. It identifies administrative domains as working environment for dedicated business roles, Administrative domains are separated from each other. The communication between administrative domains is done via well-defined reference points. A reference point specification does not only contain an API or protocol specification. Moreover, a reference point is characterized by constraints, contracts, QoS requirements, and security definitions assigned to the communication exchange between two different administrative domains. Such information is also called Service Level Agreement (SLA)[20]. Contracts define the relationship between different business roles. Constraints define additional require-

[19] In this example, the Context Interpreter augments the information that has been sent by the SDO. This is because the I-centric Service processes high level abstractions of the underlying infrastructure ('to cold'). Nevertheless, an I-centric Service can subscribe directly to any ambient information provided by SDOs.

[20] Service Level Agreement (SLA) [FAIN]

ments between different administrative domains or business roles. In example, one constraint for the communication between two business roles can be the existence of a third one that acts as a trust center for their communication. Stakeholders represent the highest aggregation level. They compose different administrative domains.

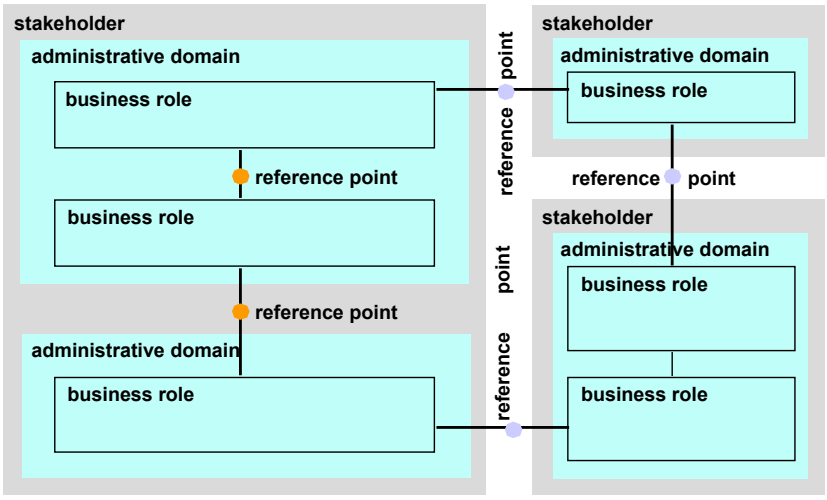


Figure 17: Generic Business Model

To illustrate the flexibility of such a business model, a short example for a business interaction is given in the following. The example uses four different business roles: customer, retailer, provider, and user.

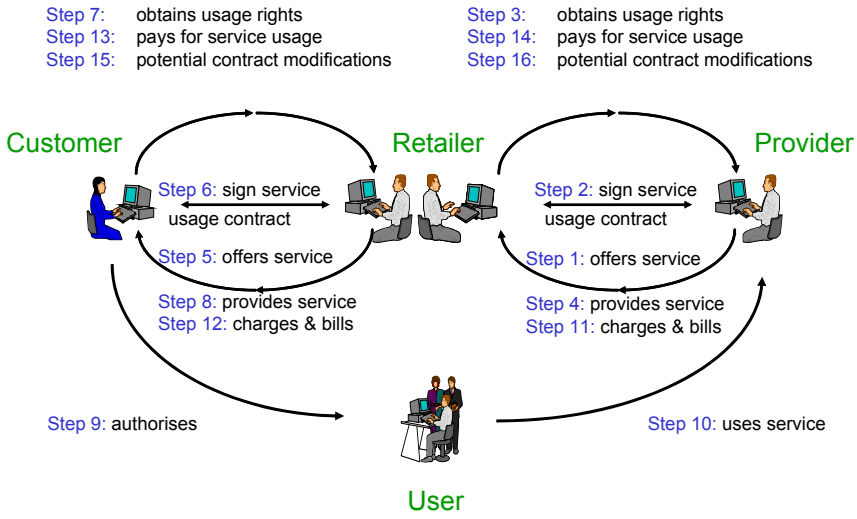


Figure 18: Examples of Business Interaction

From step 1 to step 16, the complete process of offering services at the provider side up to the service usage at user side is shown. A clear separation of responsibilities and relationships is required to ensure a trustful business interaction.

The scenarios outlined within the vision for I-centric communications require for exactly such kind of interaction by adding again the individual user as the central element. Individuals will act in distributed ad-hoc environments changing dynamically roles providing and consuming information form other individuals.

3.2.2.1 I-centric Implications for Business Modeling

As stated before, the influence of business models on functional architectures will increase. Rather than to be the result of a research and development roadmap towards a coherent system, the I-centric communications system will be made up of various technologies configured to

specific user requirements as identified in the market. One of the most difficult problems for information and communication technology suppliers is linking product capabilities to evolving demand in the market. Historically, this has led to a complex relationship between the research and development, and financial sides of companies that provide information technology.

The problem is now even more complex. Much of the technology development process is now distributed among many companies and some major players in key information and communication technology segments are downsizing their research and development commitments.

Information and communication technology now typically involves a substantial design component in which various technologies from different suppliers are configured together. Relationships between the installed technology base and the planning of future systems, products, and services were once managed via non-proprietary standards that were linked to the research and development cycles of dominant firms in various information and IT industry segments.

As the supply environment becomes much more fluid and dynamic, the design function can assume much of this co-ordination role, potentially creating new markets for new technologies.

How this design of future systems and services will materialize is not yet clear. An important research question should be raised here. How are the reference points being affected by the business environment? In order to answer such questions, dynamic modeling techniques are required. [W-WG2-BM]

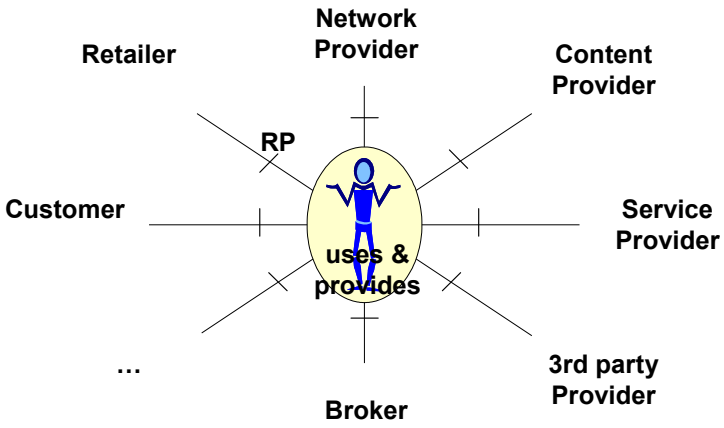


Figure 19: Individual as Center of the I-centric business model

The first major research task is the development of a generic business model for I-centric communications. This generic business model contains a set of common terminology as well as a generic framework of Business Roles, Stakeholders, Administrative Domains, Relations, Reference Points, and Revenue streams. The individual can act as any available role communications with any other role (see Figure 19).

The generic model has to support the vision for I-centric communications featuring ambient-awareness, personalization, and adaptability.

The function of the model is to:

- specify a common frame for all roles in a system
- allow flexibility against business, regulation and technical change
- allow to define the Reference Points in a multi-domain environment
- allow deriving requirements for systems of different stakeholders

Again, the generic business model introduced in Figure 17 gives enough flexibility to be applied to I-centric communications. The remaining question is: How to integrate the individual user and the dynamic character, due to changing business roles, in such a model.

3.2.2.2 Relations between I-centric Communication Spaces

The vision for I-centric communications has introduced the concept of individual communication spaces. An individual user is interaction with objects in his communication space. Communication between different individuals is done by sharing objects of their individual communication spaces as shown in Figure 20.

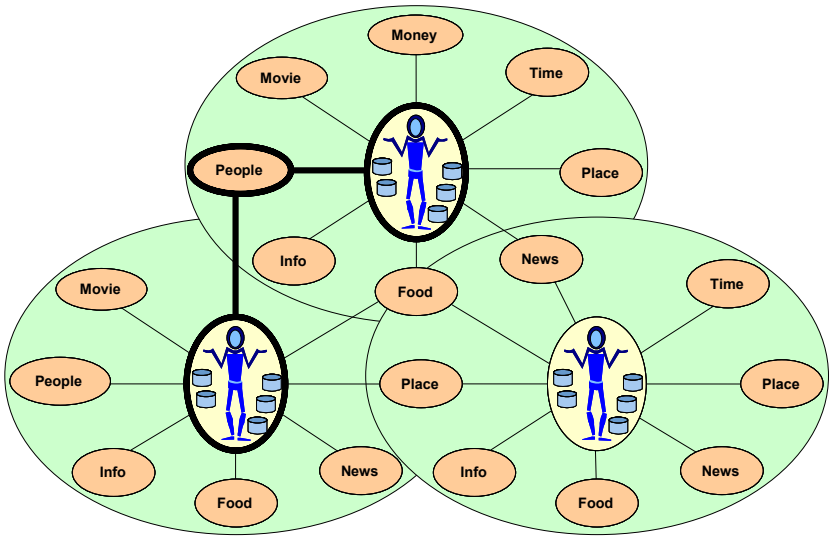


Figure 20: Interconnected Communication Spaces

The particular physical resources, which are used for a certain communication request, are determined in the activation process of contexts and objects. These physical resources belong to administrative domains where certain roles are assigned. As users act in different contexts, the relationships to the administrative domains involved in a communication process have to be managed.

This causes the dynamic assignment of roles and employment of different reference points from the viewpoint of an individual. This fact is also amplified, as the environments themselves are highly dynamic and characterized by ad-hoc and peer-to-peer communication may be without any centralized control.

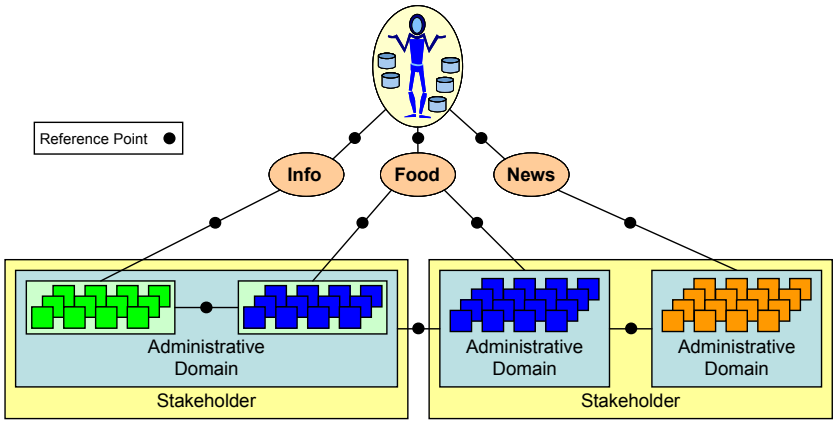


Figure 21: Mapping Objects to Business Relationships

The dynamic relationships between individual users and objects require new concepts like online-subscription and accounting, micro-payment, and federation between ad-hoc communication environments.

The temporal unavailability of objects and services, and the question: who pays whom for providing or using any physical resource is to be answered. A promising approach is to specify reference points between all involved instances, like introduced in the [W-TINA-BM].

The reference points have to support the functions requested above. For example, micro-payment between two objects has to be reflected by specifying the information to be exchanged and the relationships to other objects, which might act as certifying instances, billing, or accounting server.

3.2.3 Ontology Definitions for I-centric Communications

Section 3.2.2 has shown that a variety of actors will provide and consume different services accompanied by a continuous change of their roles and relationships to each other. To support such a flexible structure, a generic description for objects and services provided in individual communication spaces is required. Such a description will enable dynamic service (and object) discovery, online subscription, and ad-hoc interaction with beforehand unknown computing resources (object instances). [Base02]

'An *ontology* gives meanings to symbols and expressions within a given domain language. The ontology performs the function of mapping a given constant to some well-understood meaning.' [W-FIPA]

Objects, as introduced in the vision for I-centric communications, *provide well-defined services from the perspective of an individual* (see section 2.2). Well-defined in this sense is two-fold. On one hand, it refers to the technical syntax to describe the services an object provides and the rules combined with it. Secondly, it refers to the semantic part that describes the meaning of what objects are capable to do. [Nagao94]

Ideas, how this can be achieved are found in the research area of intelligent agents for instance. The *Foundation for Intelligent Physical Agents* (FIPA) has developed a specification for an *ontology service,* which enables agents to manage explicit, declaratively represented ontologies [W-AOS, Per]. Such a service can be used to declare the semantic of a certain categorization through a simple reference to the appropriate ontology description offered by an ontology service agent.

The basic approach, to describe objects, is furthermore backed by activities undertaken in the area of Knowledge Based Systems and Artificial Intelligence. In the following, the general ontology concepts that have been applied to the vision of I-centric communications are explained.

3.2.3.1 Background

The terms 'data', 'information', and 'knowledge' are often used synonymously in colloquial language. These terms have to be differentiated to describe the functionality of the services that are provided by objects in an individual communication space. In the following, these terms are used as described below [MS]:

- Data is an arbitrary, informal piece of information without explicit structure or format.
- Information is understood as well represented data with a signature.
- Knowledge is information together with interpretation rules.

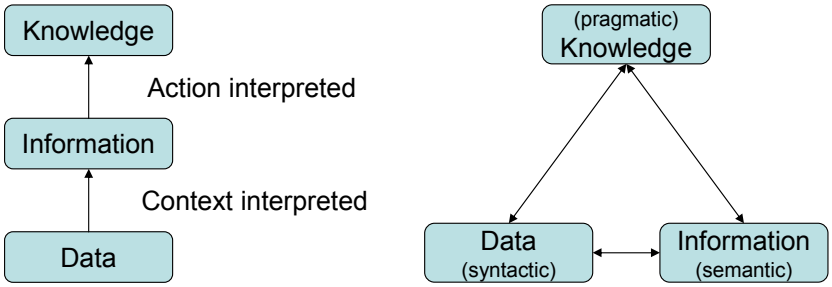


Figure 22: Knowledge Evolution & Semiotic Triangle

Data is also associated with syntax, information corresponds to semantic, and knowledge takes the pragmatic part. [MS] Figure 22 shows the semiotic correspondence of data, information, and

knowledge. In addition, Figure 22 shows the evolution of data, information, and knowledge. Data forms the basis for information. If data is evaluated in a context, then data gets a meaning and becomes to information. Knowledge is extracted from interpreted information.

Figure 23 shows a distinction into different knowledge levels. The knowledge levels are similar to interpret as the evolution steps in Figure 22. A new level on top of knowledge is called meta-knowledge. Meta-Knowledge is knowledge about knowledge, for instance structural or strategic knowledge of a domain.

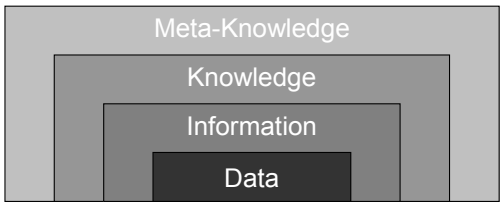


Figure 23: Knowledge Levels

The border between data, information, knowledge, and meta-knowledge is not fixed. The classification depends on viewpoint and background. The following example [MS] shows the correlation between data, information, and knowledge.

Example – digital pictures: While on the data level only bit streams are represented, the information level may contain additional format descriptions (especially those that identify the data as being a picture). Different information may be derived from the same data. On the knowledge level there may be semantic descriptors identifying the type of the picture (e.g. a landscape). Searching for landscape pictures in a database would have no result. The information system may select pictures from the database and only on the knowledge level, a landscape painting could be distinguished from a portrait.

Knowledge Engineering

As it was described in the example above, data, information and knowledge are represented in different forms. Data is represented as a bit stream or character stream. Information is represented as the data stream and its structure. Research in the area of Artificial Intelligence tried to find descriptions for knowledge. It was attempted to build systems, which contain and process knowledge, called Knowledge-based Systems (KBS) [W-KBS], e.g. expert systems. The first approaches attempted the knowledge of experts to be transferred into rules and data structure. The beginning success from small academic prototypes was not transferable into a large commercial system. Knowledge Engineering shifted the paradigm from the ‘transfer approach’ to the ‘modeling approach’. Knowledge Engineering ‘turned the process of constructing KBSs from an art into engineering discipline’ [SBF, Hei]. The change of the paradigm has the following consequences [SBF]:

- like every model, such a model is only an approximation of the reality
- the modeling process is a cyclic process
- the modeling dependents on the subjective interpretations of the knowledge engineer

Some outcomes of Knowledge Engineering are described in the next sections.

Principles

A set of design criteria for ontologies are described in the following. They are used to develop ontologies. This list was extracted from the articles [Per, Grub] by A.G. Perez.

- **Clarity and Objectivity**, the ontology should provide the meaning of defined terms by providing objective definitions and also natural language documentation
- **Completeness**, a definition expressed by a necessary and sufficient conditions is preferred over a partial definition
- **Coherence**, to permit inferences that are consistent with the definitions.

- **Maximize monotonic extendibility**, new general or specialized terms should be included in the ontology in such a way as does not require the revision of existing definitions.
- **Minimal ontological commitments**, making as few claims as possible about the world being modeled, which means that the ontology should specify as little as possible about the meaning of its terms, giving the parties committed to the ontology freedom to specialize and instantiate the ontology as required.
- **Ontological Distinction Principle** [Borgo], which means that classes in an ontology should be disjoint. The criterion used to isolate the core of properties considered invariant for an instance of a class is called the Identity Criterion.
- **Modularity** [Bernaras], minimize coupling between modules.
- **Minimize the semantic distance** between sibling concepts [Bernaras]. Similar concepts are grouped and represented as subclasses of one class and should be defined using the same primitives, whereas concepts, which are less similar, are represented further apart in the hierarchy.
- **Standardize names** whenever is possible

Types

Types differentiate in the amount and type of structure in the subject of the conceptualization. The first dimension has three categories, which can be seen as different quality levels of ontologies. Quality means the data, which is represented of it, e.g. data, information, and knowledge. [Hei]

- **Terminological ontologies** such as lexicons specify the terms that are used to represent knowledge in the domain of discourse.
- **Information ontologies**, which specify the record, structure of databases. Conceptual schemata of databases are an example of this class of ontologies.
- **Knowledge modeling ontologies** specify conceptualizations of the knowledge. Compared to information ontologies knowledge modeling ontologies usually have a richer internal structure.

The second dimension has four categories. It differentiates the subjects of the conceptualization.

- **Application ontologies** contain all the definitions that are needed to model the knowledge required for a particular application. Typically, application ontologies are a mix of concepts that are taken from domain ontologies and from generic ontologies (which are described below). Moreover, application ontologies may contain method- and task specific extensions. Application ontologies are not reusable themselves. They may be obtained by selecting theories from the ontology library, which are then fine tuned for the particular application.
- **Domain ontologies** express conceptualizations that are specific for particular domains. Whereas the domain knowledge describes factual situations in a certain domain (e.g. chest pain is a manifestation of atherosclerosis), the domain ontology puts constraints on the structure and contents of domain knowledge (e.g. diseases have findings as manifestations).
- **Generic ontologies** are similar to domain ontologies, but the concepts that they define are considered generic across many fields. Typically, generic ontologies define concepts like state, event, process, action, component etc. The concepts in domain ontologies are often defined as specializations of concepts in generic ontologies.
- **Representation ontologies** explicate the conceptualizations that underlie knowledge representation formalisms. They are intended to be neutral with respect to world entities. That is, they provide a representational framework without making claims about the world. Domain ontologies and generic ontologies are described using the primitives provided by representation ontologies.

The classification of an ontology, which is used to utilize knowledge for a system, can be helpful to find a suitable description language respectively description format.

Components

Knowledge in ontologies is formalized using four kinds of components: classes, relations, functions, and instances.

- **Classes** are used in a broad sense. They can be abstract or concrete, elementary (electron) or composite (atom), real or fictitious. A concept can be anything, about something is said and, therefore, could be the description of a task, function, action, strategy, reasoning process, etc.
- **Relations** represent a type of interaction between concepts of the domain. They are formally defined as any subset of a product of n sets, that is: R : C1xC2x ... xCn. Examples of binary relations are *subclass-of* and *connected to*.
- **Functions** are a special case of relations in which the n-th element of the relationship is unique for the n-1 preceding elements. Formally, functions are defined as: F : C1xC2x ... xCn-1 → Cn. Examples of binary functions are *Mother-of* and *square*, and an example of a ternary function is *price-of-a-used-car* that calculates the price of a second-hand car depending on the car-model, manufacturing date and number of driven kilometers.
- **Instances** are used to represent elements.

Components explained here, are similar to them, which are used in the object-oriented design and in the entity relationship modeling. That is not surprising, because they are also used to reflect a part of the reality (existing knowledge).

3.2.3.2 Applying Ontology to I-centric Communications Systems

The Open Profiling Framework, as introduced in section 3.2.1, requires many interactions between the components of an I-centric communications system. Information about object states, ambient information, preferences, and instructions for objects to be executed have to be exchanged.

To ensure a formal correct processing of all this information, this section introduces a general structure for ontology. All information that is exchanged within the Open Profiling Framework has to be compliant to the ontology structure introduced in the following sections.

Ontology tasks

From an I-centric perspective, the ontology describes the functionality that is offered by the objects of an individual communication space. Ontologies combine the semantic behind the functions objects provide with signature and syntax to activated the objects. Regarding the background discussion of ontologies (see section 3.2.3.1), the ontology structure defined in the following sections represents a 'generic ontology' as it can be applied to many different application fields.

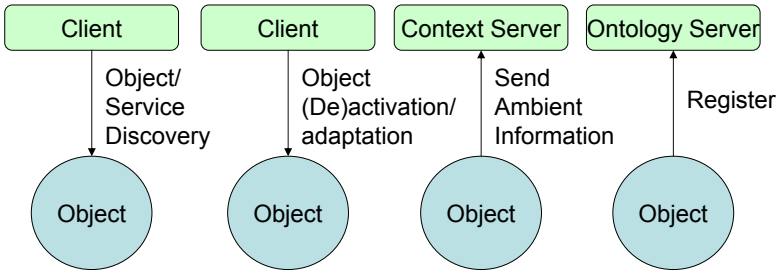


Figure 24: Simple use cases for Ontology usage

Figure 24 shows the use cases where ontologies are used in I-centric communications systems:

- **object / service discovery**: to resolve, which services an object provides, and to resolve, which parameters are needed to control a certain service
- **object (de)activation/adaptation**: to instruct objects how they should behave
- **sent ambient information**: to collect ambient information that is compliant to ontology

- **register**: to introduce an object to an I-centric communications system
- **service creation and deployment**: to generate and install I-centric Services
- **ontology mapping**: to map between information compliant to different ontologies (semantics of information is the same but syntax or vocabulary differs)

Object / service discovery providing the retrieval of object's ontology to be evaluated by the client that initiates the retrieval. A client in this case can be any other component inside the I-centric communications system, including other objects. Especially 'object-object' discovery enables the ad-hoc instantiation of autonomous groups of objects that dynamically come together to solve dedicated tasks. In this case, an object discovers the functions provided by another object to find out whether that object can be used to fulfill its own task. The discovery mechanism has to return a unique ontology identifier or the object's functional signature (operations and parameters) to be further analyzed.

Object (de)activation and **adaptation** require the same information as the discovery functions. In addition, an (de)activation process has to generate an instruction to be send to an object. That means the ontology of the object to be activated must be known to the client beforehand (the client can retrieve it via the discovery function). The object itself evaluates each instruction it receives against its own ontology to find out whether it can handle the request. If the result of that evaluation confirms the compliance to the own ontology, the instructions are further analyzed by the object. This mechanism prevents any object from processing a request it is not able to 'understand'. Usually, the clients for object activation or adaptation are I-centric Services. Interactive user applications, directly interacting with objects within a communication space, can also be clients in this case.

Comparing the requirements, identified so far, with traditional ways to specify interfaces of distributed objects, a major difference should be emphasized here. I-centric objects have to provide a more precise definition of operations and parameters to prevent the processing of invalid instructions. Interface definition languages like IDL[21] and ODL[22] provide the definition of data structures to be used in the interface only. The ontology description for I-centric communications has also to provide a description what values are allowed within certain data structures (e.g. a range of valid attributes).

Sending ambient information from an object to the Context Server requires the same behavior of the Context Server as before explained for the object receiving activation. The Context Server must know the ontology of the sending object to handle the received information in an appropriate way. If the Context Server receives any ambient information referring to an unknown ontology it just denies the processing of that information.

Register is the function used to introduce new objects/ontologies to the I-centric system. Once registered with the Ontology Server, an object can start to sending ambient information. The registration ensures the uniqueness of ontologies by analyzing them during the registration process. All registered ontologies have a unique ontology ID, which is issued by the Ontology Server. If an object tries to register with an already registered ontology, the registration will be confirmed to the object by returning the already registered ontology ID.

---

[21] Interface Definition Language [OMG-IDL] – enables the definition of interfaces, regarding operations and parameters by specifying the contained data structures. An IDL is a purely a descriptive language whose interface provides the information needed to develop clients that use the interface's operations.

[22] Object Definition Language [Z.130] – does the same then IDL by adding textual constraints to the interface definitions. Constraints cannot be evaluated automatically. Only the programmer who is going to implement a certain object might take such constraints into account. ITU-ODL supports features that are not covered by OMG-IDL. The new features include the computational language of the Reference Model for Open Distributed processing [X.901-904] and include multiple interface object templates, group templates, stream interface templates and Quality of Service (QoS) descriptions.

The use cases shown in Figure 24 represent simple interactions, in which ontology is involved. Figure 25 depicts more complex use cases that require several interaction steps between components. During Service Creation & Deployment, several interactions between the Service Builder and the Context Server/Ontology Server are needed.

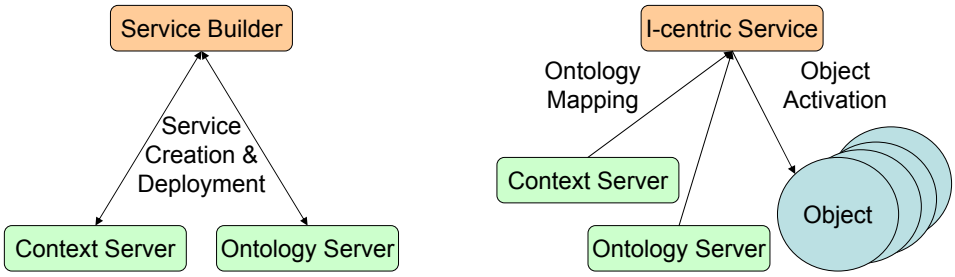


Figure 25: Complex use cases for Ontology usage

An I-centric Service, designed to activate a certain set of objects (knowing about their ontologies), can utilize the **ontology mapping** to control objects that have different ontologies. The ontology definition for I-centric communications therefore has to provide a mapping mechanism that links different ontologies together. Mapping different ontologies has to be possible in two ways. The first way assigns an operation specified in the one ontology to another operation specified in another ontology, meaning that both ontologies support the same operation. Figure 26 shows an example where two different ontologies are given providing the same operation (operation name, operation parameters, and operation semantic are identically). The link between the two ontologies defines that both ontologies are semantically and syntactically the same. If such a link has to be established the creator (a human system operator) of that very link has to check the equivalence of semantic.

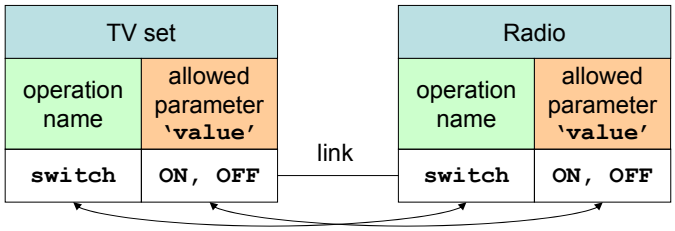


Figure 26: Linking ontologies

The example given above enables mappings between a TV set and a Radio. Both can be switched ‘on’ and ‘off’. A service designed to control one of them (TV set or Radio) can now control both checking whether an ontology provides a link to another ontology.

The second way of linking ontologies provides a mapping function between the parameters of different operations specified in separated ontologies. **Mapping ontologies** can be applied when two or more ontologies provide the same functionality (semantically) by requiring different syntax to parameterize operations.

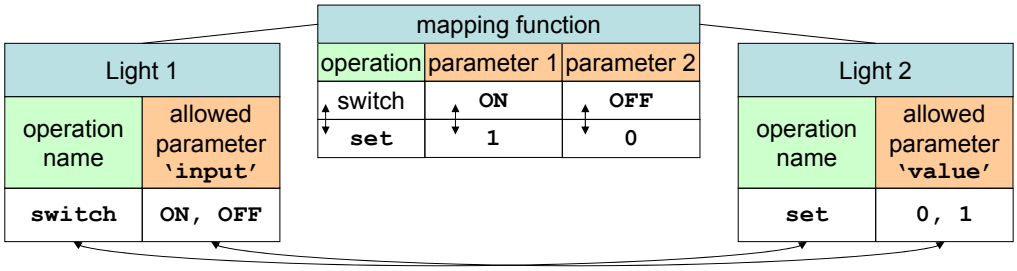


Figure 27: Mapping Ontologies

As for the other example, the establishment of such a mapping can only be a process where a human operator is involved. It has to be determined which operation is semantically equal to another and how the mapping between the parameter has to be processed. Figure 27 gives a

simple example how to map between ontologies of two different light objects. In this case, the parameters are just a short list of numbers or strings. Corresponding 'allowed parameter values' are assigned to each other.

The mapping function enables I-centric Services designed for activating objects of type 'Light 1' to activate objects of type 'Light 2' too, and vice versa. In general, a mapping function has to provide possibilities such as:

- mapping between different numbers of parameters,
- exchange of parameters or data types, and
- arithmetic translation of parameters.

Another problem, which can be solved by mapping ontology, is caused by evolutionary characteristic of software-based systems. An object once developed may be altered to improve its functionality. Typically, that means the interface of that object is altered and cannot be used at all without modifying all clients too. Using ontology mapping in such a case can ease the usage of a modified object by mapping the 'old' operation to the 'new' one without modifying 'old' clients.

The functions introduced above have to be reflected by the ontology used in I-centric communications systems. The next section introduces the proposed structure of the ontology.

3.2.3.3 General Ontology Structure for I-centric communications systems

The general ontology structure consists of four parts, namely GENERAL, CONTENT, USAGE, MAPPINGS, and DESCRIPTIONS as shown in Figure 28. These parts reflect all necessities that have been identified and described above.

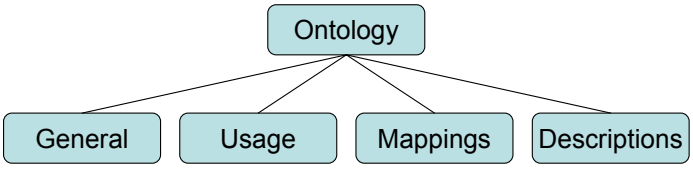


Figure 28: General Ontology Structure

The **GENERAL part** of the ontology is designed to identify ontology. Therefore, GENERAL contains three different entities as depicted in Figure 29. The TYPE determines what kind of information is described by ontology. Predefined by the vision for I-centric communications, possible values for the TYPE are object, context, active context, preference, ambient information, and I-centric Service.

NAME gives the ontology a unique name that is used as reference to certain ontology. An I-centric communications system must provide a mechanism that ensures uniqueness of names. VERSION allows the management of different versions of ontologies. If the functionality of an object has been improved, this object can be registered again within the I-centric system providing then a new ontology. The new ontology accords to the improved (new) functionality. The new ontology indicates the relationship to the old one by increasing the 'old' version number. The 'new' ontology can only be understood by new clients. An ontology mapping has to be provided to support 'old' clients.

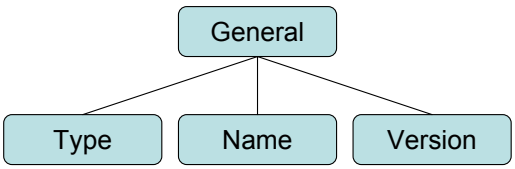


Figure 29: General Part of the Ontology

The **USAGE part** of the ontology describes the operational interface of objects. The structure of the USAGE part is shown in Figure 30. Operations are defined by giving them an **operation**

name, an **input parameter(s)**, and an **output parameter(s)**. That reflects the needed information to describe procedural interfaces as any object oriented programming language does. The input parameter and output parameter can be described as Interface Definition Languages do by specifying the data structures they contain.

In addition, all allowed values that these data structure can contain have to be specified. That can be done using a constraint language like [W-OCL] or a mechanism to specify data type restrictions as introduced in the XML Schema specification [XSD]. The restrictions or constraints have to be incorporated with the definition of the data structures.

An example using pseudo syntax is given below to illustrate USAGE.

```
USAGE of TVset
   Operations
     Operationname = 'switch'
       InParameter (string) value restricted to 'ON|OFF'
       OutParameter = non
       Link = 'Radio'
```

A client receiving a USAGE part as given above has to interpret the given information. Based on the evaluation, the client is able to construct commands to be sent to the object compliant to the given ontology and the respective USAGE part (e.g. `switch(ON)`). LINK is referring to an operation having the same syntax and semantic, which is part of another ontology as described in section 3.2.3.2 (in the example LINK is referring to the Radio ontology function 'switch').

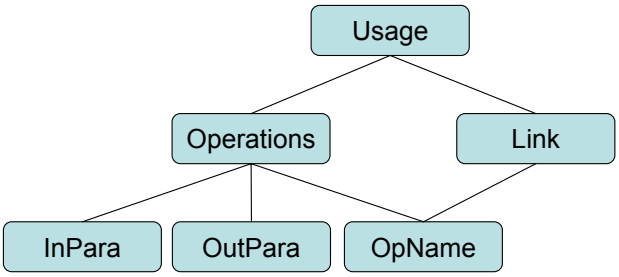


Figure 30: Usage Part of the Ontology

The **MAPPINGS part** defines what mappings are possible between the USAGES part of different ontology instances. MAPPINGS identify a TARGET ontology (TARGET) and specifies which operations can be mapped to the target ontology. The MapFunction defines an algorithm, which has to be applied to the In-Parameters and Out-Parameters to be compliant to the target ontology.

Figure 31 shows the structure of the MAPPINGS part of an ontology. Below a concrete example of how to describe a mapping between two ontologies is given. Pseudo syntax is used again to show which parameter of the source ontology has to be mapped to the target ontology.

```
MAPPINGS of Light1
  Target = set of Light2
  InPara
    SourcePara = input
    TargetPara = value
    MapFunction: input(ON) = value (1),  Input(OFF) = value (0)
  OutPara
    non
```

The mapping above illustrates the example given in Figure 27 again. A client able to issue commands to Light1 (e.g. `switch(on)`) can have this command mapped according to the ontology of Light2 (e.g. `switch(on)` is mapped to `set(1)`).

The exact specification of the MAPPINGS part depends on the selected technology to describe the ontology. Therefore, a complete specification is given in the realization part of this thesis (see section 4.3.2).

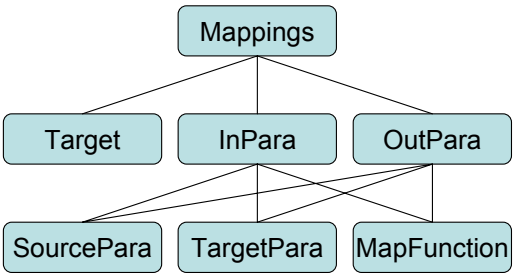


Figure 31: Mapping Part of the Ontology

The DESCRIPTION part of the ontology describes the semantic of an ontology and provides several types of notations for that. To support the developer of comfortable graphical user interfaces, the DESCRIPTION part first defines the human language a description is given in. Descriptions can be given in different languages in parallel to support multilingual services or user interfaces.

Furthermore, the DESCRIPTION part provides textual descriptions as well as multimedia content to be used to visualize the semantic of the ontology. All information provided within the DESCRIPTION part is to assist human users to understand the purpose of a discovered object. Object-to-object interaction will not evaluate the DESCRIPTION part at all.

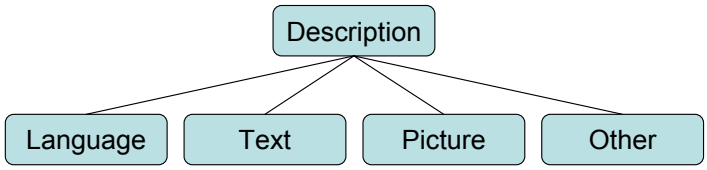


Figure 32: Description Part of the Ontology

The ontology structure introduced so far enables the description, discovery, validation, and mapping of interface descriptions provided for objects within individual communication spaces. In addition to that, the ontology structure should also suit the knowledge or the information handled within I-centric communications systems (e.g. to describe individual preferences or content to be exchanged between different individuals).

For such situations, a modified ontology structure can be used. The USAGE part has to be exchanged by a simple INFORMATION part, which define the data structure and restrictions for the information to be exchanged. The same mechanism as introduced for InPara/OutPara can be applied then. An INFORMATION part can also be linked or mapped to another ontology. Linking INFORMATION indicates that two pieces of information are semantically and syntactically same. Mapping INFORMATION part describes the translation between the information stored in different profiles.

An example for mapping between profile info is given in the following. An individual might have the preference to get graphical user interfaces displayed with blue background color. A generic user profile then will contain the preference ‘blue’. Whereas a more hardware related profile will store the same information as ‘RGB(50,10,200)’. The mapping between both profiles enables a checking algorithm that finds out whether both representations are semantically same.

The introduction of using ontologies for specifying the interfaces and information to be exchanged between different components of the Open Profiling Framework leads to a flexible infrastructure. Components can be discovered automatically, exchanged information can be evaluated against its ontology, and the process of software maintenance is eased by reducing the programming overhead when some of the components change.

3.2.4 Personalization

For the provision of I-centric Services, it is necessary to cope with the management (gathering, categorizing, storing, updating, and erasing) of preferences within an individual communication space and the evaluation (interpretation of contexts) of that information. The process enabling service personalization is called **profiling**. It is obvious that these properties have to be described and stored in profiles, facing the introduced dimensions of contexts in I-centric communications.

3.2.4.1 Background

Technically seen, profiles are data structures, which have to be stored persistently. Profiles can be created by individuals manually, by machine-machine interaction, or by self-learning capabilities automatically. Profiles store information and knowledge as introduced in section 3.2.3. That comprises the semantic a service combines with a profile. Ideas, how such profiles could be defined, have been developed in the research area of agent communication languages [W-ACL], which define ontologies of data structures.

Beside 'pure' storing technologies, some other problems have to be taken into account. I-centric communications systems have to have a profound knowledge about individuals for realizing the required functionality. That knowledge comprises sensitive information to be taken into account by the system, but kept inaccessible for other individuals (i.e. content description of information to be delivered). Furthermore, the information/personal data present in the profiles needs to be accessible by different stakeholders/interested parties with a different amount of detail. This will require multiple techniques, like password protection, encryption, user-defined authorization, but is also partly covered by legislation. On the other hand, such kind of information has to be partially accessible for a trusted group of individuals. To solve these problems, such a system has to provide trust / security mechanisms, which ensure user defined policies for information access like defined in [W-P3P].

However, the actual representation of an individual's context in data structures is an important issue of profiling, because it influences not only the structure of profiles and the data storage but also the realization of profile evaluation. Since the profile evaluation method has to be able to process the created profile structures, it is directly bound to the actual chosen profile-structuring methodology. Therefore, choosing the appropriate standards/technologies/products for profiling comprises two steps:

- choosing a **profile-structuring** methodology that is open and flexible
- realizing **profile evaluation** able to cope with the specified profile structure

The general processing of profile evaluation is explained in section 3.2.6 (Adaptability). Since Adaptability has to consider both preferences and ambient information, which are stored in profiles, a general profile evaluation mechanism has been developed and applied to the idea of service adaptability.

Managing Preferences

Preference information of an individual consists of several different properties representing objects and contexts of the individual communication space.

All information that can be gathered within an individual communication space about individuals and objects has to be described and stored in profiles. A profile is a data record that electronically represents/contains information on the context of the entity it reflects upon. As such, the whole profile information can be regarded as a set of metadata describing the individual user with his abilities and needs. Presence related information, terminal related settings, home environment specific data, application specific data, service provider specific data, user-related preferences, and location related data could be specified in the profile. Sometimes, certain parts of the profile information will be specified; sometimes they will not be present.

Different players will have different parts of the profile information. The profile information gathered, used, and stored by different players will not be accessible for the individual in general. The profile information will never be complete. Further, it will not be stored in one location. It can reside partly in the network (distributed) and partly on (multiple) terminals. It can be updated, thus synchronization of the profile information parts needs to be addressed. The structure as well as the meaning of a property in a profile can change over time. Therefore, an open profiling mechanism is needed that is flexible enough to handle changing structures of profiles. With regard to the implementation of profile date, these characteristics lead to a preferred format that supports

- semi-structured data,
- extendibility,
- distributed storage and management,
- interoperability, and
- security of privacy-sensitive parts

Furthermore, the profile information needs to be dynamically adjusted according to the changing context of the mobile user. Typical for mobile users is that they are able to move around and while moving, his context or environment can change. While moving, the physical location, but also the infrastructure like the network access can change.

Due to the distributed nature of profile (they are gathered, used, stored, and managed by multiple stakeholders and at several locations), a generic support for distributed context information is needed for a proper facilitating of service adaptation to the context. In general, the central idea of importing (parts of) different metadata standards yields so-called application profiles. Application profiles are defined as metadata schemes, which consist of metadata elements drawn from one or more namespaces (ontology), combined by implementers, and optimized for a particular local application.

The main issue is that profiles must be stored in an adequate way in an appropriate place to be able to use the information at the right moment. The profiles are the very basic set of information for personalization and adaptation of the individual's communication space. Therefore, these sets of information need to be available in a ubiquitous way.

The place to store the profiles and contexts is difficult to determine because it can be used at different places at the same time. On the one hand, it is the simplest way to store it on the mobile terminal and on the other hand, it can be located in the network for processing reasons.

Preferences and Context Representation

In order to be able to personalize the objects in individual communication spaces, it is important to know the individual preferences. These preferences can be gathered or learned in different ways and using different mechanisms.

Therefore, it is important to define the semantic meaning (ontology) of the different information items within a profile that contains preferences. The provided sets of information can be sharply defined or specified in a fuzzy way. Therefore, it is necessary to define the meaning of these information sets formally, as already discussed in the ontology section of this thesis (see 3.2.3).

Personal information and the user's profiles must be subject to privacy and security considerations. The most obvious examples for sensitive information might be a credit card number or an individual's location at a certain point in time, but also other context information such as terminal preferences, individual interests, user credentials, etc. should be considered as sensitive with respect to privacy and security. It is likely that the end users' awareness of privacy issues will soon raise to a much higher level than it is currently in the Internet world. Users will decide more carefully which service providers they trust. The current introduction of location-dependent services as well as the rapid growth in email spamming seems to become triggers for an increasing awareness of privacy concerns.

3.2.4.2 I-centric Model for Service Personalization

As discussed in the sections above, Service Personalization is mainly focused on profiling issues. What information has to be profiled is the central question for personalization in the context of I-centric communications is therefore.

Figure 33 shows the structure of information that has to be taken into account for I-centric Service personalization. This structure follows the Reference Model for I-centric Communications, where personalization is considered as information about **objects**, **contexts**, and **preferences**.

Objects

The information to be profiled about objects is characterized by the services an object offers via well-defined interfaces. The ontology of objects has to taken into account since individual users are only interested in the semantic of the provided services. An object has to provide its ontology to the I-centric system to be dynamically analyzed during the service discovery process. The general structure for ontology, as introduced in section 3.2.3.3, is used to describe the ontology of objects.

This ontology structure also contains a mechanism to describe the signature of objects (operations and parameters). Nevertheless, the signature only provides a high-level abstraction of the objects functionality. The operations and parameters as given in the ontology does not necessary describe the actual implementation of the interface technology used for an object implementation. Therefore, the physical interface that might be used to tunnel the signature given in the ontology has also to be known to the I-centric communications system.

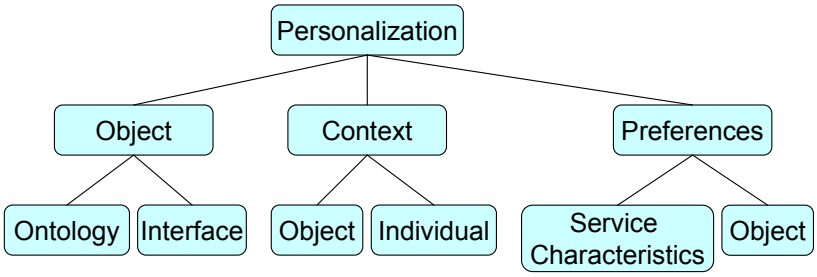


Figure 33: I-centric Model for Service Personalization

For example, the signature given in the corresponding ontology describes operations and parameters using WSDL[23] specifications. The used middleware technology to enable the interoperability between different objects might be CORBA [W-CORBA] using IDL specifications for interfaces. In this case, the middleware acts as a transport layer without interpreting operations and parameters.

For this reasons, the I-centric communications system has also to store the used interface technology of an object to provide I-centric Services with this information.

Context

The profile information about context contains the relationships between objects and between individual users and objects. From a profile point of view, a context manages a list of typed relations to a flexible number of objects and individuals. One relation to an individual represents the fact that a context belongs to that individual. Relations to more that one individual indicate a

[23] WSDL is an XML format for describing network services as a set of endpoints operating on messages containing either document-oriented or procedure-oriented information. The operations and messages are described abstractly, and then bound to a concrete network protocol and message format to define an endpoint. Related concrete endpoints are combined into abstract endpoints (services). WSDL is extensible to allow description of endpoints and their messages regardless of what message formats or network protocols are used to communicate. [W-WSDL]

user expressing the wish to include other individuals in a certain context (e.g. to define team of workers that perform a task together). The type assigned to a relation indicates the semantic or constraint an individual user wants to establish. The type has to be described within ontology to be discoverable and able to be evaluated.

For example, there might be an 'off duty' context, where an individual user has specified to include all of its friends with the constraint 'I'm reachable for them' and 'Breaking Sports News' to be received if any available. Furthermore, all business contacts could have been included, defining constraint 'not available' for them. An I-centric Service responsible for incoming communication and information would now block all business related contacts. Only friends or sports news would reach the individual. Such kind of activity needs an active context, where all involved physical objects have been resolved, activated, and configured accordingly.

Preferences

The ontology given for a dedicated object describes the services this object is offering (what kinds of operations have to be parameterized using which values). Preferences represent a selection of operations and specific parameters for a certain object. Therefore, preferences follow a given ontology of an object by specifying dedicated operations and values. Preferences can be seen as preconfigured settings ('choices of service characteristics' as described in section 2.2.2) for objects to be activated.

Example: an object might offer an operation `'switch'`, which allows the parameters `'ON'` and `'OFF'`. A preference for this object can be `switch(ON)`. If now a context is activated that contains a relation to that very object, the object will be switched on, according to given preferences.

As described in this section, the I-centric model for service personalization requests for a number of information to be profiled. The Open Profiling Framework, which has been introduced in section 3.2, defines the Context Server and Ontology Server to be responsible for managing all this information. A detailed insight in all these components and information handled by them is given in chapter 4.

3.2.5 Ambient-awareness

Ambient in the sense of I-centric systems refers to the physical and organizational environment in which an individual is. Ambient-awareness deals with sensing and exchanging ambient information related to individuals. The ambient information includes spatial, environmental, and physiological information.

Examples of spatial information are geographical data like location, orientation, speed, and acceleration. Environmental information includes temperature, air quality, and light or noise level. Physiological information can include life conditions, like blood pressure, hart rate, respiration rate, muscle activity, and tone of voice.

The goal of ambient-awareness is to acquire and utilize ambient information of an individual to support I-centric communications. In ambient-awareness, a key aspect is situation sensing where a sensing device detects environmental states of different kinds and surpasses them on.

For the provisioning of ambient-aware services, an open profiling mechanism (as introduced in section 3.2) is needed that imposes several requirements. These requirements will be summarized and worked out more clearly in the following section.

To support legacy technology as well as upcoming new devices in terms of service creation the open profiling framework can be divided into three sections:

- definition of contextual information inside profiles,
- management of these profiles, and
- profile evaluation for intelligent decision-making processes.

The definition of contexts is a fundamental issue, since it influences the management as well as the evaluation process. Later on, each part of service logic or management function has to work efficiently with the defined structures.

After gathering ambient information and preferences, the corresponding data needs to be categorized and stored. The entity responsible for the recording of information has to update the information dynamically, which includes changes in data as well as the deletion of certain parts or the data as a whole. Changes in the profile information can also mean to change the structure of a profile.

There has to be a scalable data retrieval mechanism that provides access to the gathered ambient information. This mechanism has to provide data filtering, since sensors like cameras or microphones might generate an immense amount of data. Queries to ambient information have to be offered not only for the current state, but also for all previous states, which makes it necessary to provide some kind of an ambient information history.

The interpretation and evaluation of ambient information has to be achieved using a generic information processing method. Since this topic strongly relates to research areas like artificial intelligence and knowledge based systems, it will be only slightly covered in this work. Nevertheless, a few requirements have to be mentioned:

- automatic recognition and resolution of contexts,
- reasoning and decision-making on the basis of ambient information, and
- self-learning to adapt to individual's preferences.

The Open Profiling Framework offers an open and generic approach for the usage of ambient information. The components of the Open Profiling Framework receive, categorize, store, and evaluate contextual information for the provision of I-centric Services.

3.2.5.1 Background

A variety of concepts has influenced the developments in the area of ambient-awareness. In the following, the most important developments are briefly introduced.

Ubiquitous Computing

The concept of ubiquitous computing [Abo98]

- to have ubiquitous access to computing facilities,
- to have access to any provided service, and
- the peculiarities of the underlying infrastructure and technologies disappear

requires new technologies from several areas. New kinds of wireless mobile networks up to new kinds of applications and services will characterize future ubiquitous computing environments. A significant trend, which can already be recognized on the consumer market, is the augmentation of our everyday environment. Both, the devices employed and the facilities used, are going to be augmented with sensors (e.g. GPS in cellular phones), control mechanisms (e.g. Internet-ready refrigerator, facility control), speech, and handwritten driven interfaces (e.g. Interactive Voice Response based applications, E-books). This augmentation builds the basis for ubiquitous computing. People will have nearly unlimited freedom in choosing a way to interact with a service any environment provides. [Pas99]

'… Our preliminary approach: Activate the world. Provide hundreds of wireless computing devices per person per office, of all scales (from 1" displays to wall sized). This has required new work in operating systems, user interfaces, networks, wireless, displays, and many other areas. We call our work 'ubiquitous computing' …' [W-UbiCom]

The initial incarnation of ubiquitous computing was in the form of 'tabs', 'pads', and 'boards' built at Xerox PARC, 1988-1994. Ubiquitous Computing has roots in many aspects of computing. In its current form, it was first articulated by Mark Weiser in 1988.

Pervasive Computing

Pervasive computing aims at increasing people's productivity and simplifying their everyday chores at work and at home (and even in between), anywhere and anytime, outdoors and indoors, by availing, whenever needed, the desired information to one's personal digital device.

Pervasive computing is focused on the following areas [W-Perv1, W-Perv2]:

- link: low cost, wireless link technologies, primarily Bluetooth, an open industry specification for short range RF-based communications
- network: networking-in-the-small: developing self-contained communication modules allowing a number of diverse personal computing devices to become members of spontaneously created, self-organized, pervasive computing communities
- user experience: develop technologies that would allow users to securely interact focusing only on the services they would like to exploit rather than how to find them, connect them, and access them.

Pervasive computing envisions a future in which computation becomes part of the environment. The computer forms (workstation, personal computer, personal digital assistant, game player) through which we now relate to computation will occupy only a small niche in this new computational world. When computation becomes part of the environment, most human-computer interaction will be implicit.

Context-Aware Computing

Context-aware computing was discussed by Schilit and Theimer in 1994 to be software that 'adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time'. This definition obviously tends toward the previous introduced definition of context and context-categories. It is therefore reasonable to relate a description for context-awareness to the definition of context. [Lamm2k, Ljun2k, Pana2k, Pen99]

Many researchers have defined context-awareness or rather context-aware applications in their recent works. Most of them stated that applications to be context-aware, which adapt their behavior based on context dynamically.

They require their context-aware system to detect, interpret, and respond to context. The work of Dey and Abowd [Abo99, Dey2ka], who have chosen a more general definition in their work, is described as follows: 'A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user's task.'

The features of context-aware applications are as vague as the definition itself, since the field of usage is extremely wide. To summarize the proposed features of context-awareness it can be said that it is intended to provide intelligent, customizable, situation-dependent, and self-adapting systems for application domains like communication, information, etc.

Location Aware Systems

Location aware systems combine technologies of different areas – access to fixed and mobile networks, support of multiple communications and information services, and the assistance of the user with different interfaces, e.g. graphical interfaces or speech-driven interfaces. [Har94, War97] The basic research task of location aware systems is the development of location-aware applications, which assist the nomadic user with his mobile communication equipment at his actual location. [Zel98]

Location-aware applications must have knowledge about the communication capabilities at the user's location in order to provide an appropriate service conversion in case of delivery or service access. The communication system has to adapt services based on that knowledge.

Ambient-awareness in the Scope of Telecommunications

This section introduces one of the specific application domains for the use of ambient information, namely Telecommunications. Nowadays, advanced communication systems combine classical telecommunication services like telephony or fax with data communication services.

Several applications in the wide field of communication technologies, applications, and services can be enhanced using ambient information. Therefore, two examples will be given that illustrate the role of context in the area of telecommunication.

The first example is concerned with a problem that can be described as 'Internet effect'. A typical user has access to several useful information services (provided at Internet portals) that can be personalized to his individual demands. Each of them has to be personalized in a specific, to any other incompatible way. [Arb97] Leaving for vacation for instance, a user has to switch on his answering machine at home, configure the automatic e-mail response, and redirect his telephone at work to his colleague.

In this example, ambient information can be used to provide a unified model to adapt services. The user resides in a context that states him on vacation and unreachable. Processing this context, each service can initiate appropriate actions in case of incoming communication requests. They are able to use information from contexts related to the user, like the telephone number of his colleague at work or the addresses of his personal e-mail accounts.

The second example is related to business environments. Each participant of a meeting should switch off, or mute his personal communication device, in order to avoid disturbance. A telecommunication service that uses ambient information is able to detect such a situation in which the user does not want to be interrupted and can refuse or redirect this request. It is therefore capable of reacting appropriate to the user's current communication context.

These two examples just give a first glance at the usage of ambient information in telecommunication applications.

Sensing the Physical Environment

A device is ambient-aware if it can respond to certain situations and stimuli in its environment, representing individuals and objects of the individual's current communication space. *Sensors* or *human-machine-interfaces* can gather information about these individuals and objects. *Individuals* can also provide this information themselves. Sensors can be located in the network, in devices or in the environment of the individual. Automatic gathering of ambient information is preferable form the viewpoint of an individual. Advances in sensor technology are needed to reach further adaptation of services to and co-operation with the environment of individuals.

3.2.5.2 Crunching Ambient Information

In the previous section, a few examples of sensors were given. The sensing equipment itself can be quite small and distributed. The problem might be to acquire the sensor data at an appropriate aggregation level. Knowing that the temperature varied 1 degree at 1000000 positions may not be appropriate, if the values cannot be aggregated to describe the area in relation to a known reference, e.g. the geographic location and time of the sensor; these in turn have to be aggregated appropriately. Knowing that the temperature has risen up to 15 °C in morning might indicate to people that spring is approaching, for instance.

Sensors also interact, as the example above shows. While a clock is strictly speaking not a sensor (since it does not retrieve its input from the outside, but from the oscillations of a quartz crystal), it is likely that the more general-purpose sensors become, the more they will trust other information sources. Today, if a thermometer is to communicate temperature readings, the location has to be determined and communicated out-of-band (i.e. by setting it in software or the thermometer itself). A general-purpose thermometer could sense its own location (and, using GPS, the time), and use this in communicating its values.

The above examples show that a rightful interpretation of sensor-readings is needed. Ambient information is useless, until an evaluation takes place, which can take advantage of the additional situational information. Evaluation of the individual's situation in the communication space is therefore as important to an I-centric communications system as sensing itself.

Different parts of the individual's situations (and of the context of the information, such as publisher-created document information) may have different weight in determining the ambient information of the individual. In addition, the ordering of the processing may affect the resulting ambient-awareness. In traditional programming languages (including script languages and transformation templates), there is an inherent ordering of the processing, whether consciously created or not. This ordering can be formally expressed, and externalized from the program, which actually executes the processing.

If the information about the relative weights and the ordering of processing are not communicated themselves, it has to be communicated as a separate description, essentially a meta-profile for the processing. This can be expressed in a formal rule-language, or as an ontology using the names and/or properties of the profiles. This, in turn, implies that the processing entity can handle the reception of such meta-profiles. The processing order and the relative weights would be determined by this. Database selection, trans-coding, annotation, and other contextual processes could be applied as part of this process.

Communicating and aggregating the values also present a number of special challenges. For some applications (e.g. 'if the day is Saturday, the month is August, the time is morning, the temperature is above 20 degrees centigrade, and the sun is shining, show me the way to the beach'), the number of values is quite limited, and the aggregation and rule system quite straightforward. However, if there are a large number of sensors, statistical methods may have to be used to determine the current values for the current position ('what is the temperature near me' can be determined by averaging a number of fixed sensors on buildings, for instance).

The problem then becomes one of communicating these readings, as well as one of privacy. The fact that the sun is shining is well known to anyone in the area, but the fact that a person is stressed may not be immediately obvious, and may be something that person desires to hide from others. This will require privacy protection.

3.2.5.3 I-centric Model for Ambient-Awareness

As discussed in this section, the main tasks for ambient-awareness are gathering and crunching of ambient information to be processed further by I-centric Services. Figure 34 shows the general model for ambient-awareness in the context of I-centric communications.

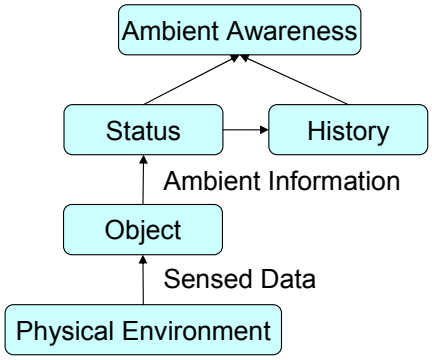


Figure 34: I-centric Model for Ambient-awareness

Individual users act in their communication spaces, which contain objects representing resources within the physical environment. The physical environment of future communication environments is expected to provide various sensing facilities to detect changes in the physical environment. Furthermore, user interfaces are considered as part of the physical environment as they are communicating with individual users directly. Objects, representing dedicated facilities

in the physical environment, receive the detected data. All information that is exchanged within an I-centric communications system has to be well defined (by means of syntax and semantic). Therefore, an object receiving any data transforms this data into information by eventually trans-coding the data and assigning ontology to the information. Such augmented data (now called ambient information) is sent to the other components of the I-centric communications system. In addition, intelligent filters can be applied by the object to the received data to notify only important changes in the physical environment to the other components of the I-centric communications system (identify individual preferences).

For example, the physical environment contains a temperature sensor. The sensor emits every fife nanoseconds the actual senses value to the object. I.e., this value is a number given in ampere. On one hand, the object receiving the values translates them into °C and discards any new value that does not indicate a significant change to the former one. The translation to °C is combined with the provisioning of an ontology that describes temperature values in °C. The mechanism that discards values prevents the I-centric communications system from evaluating unnecessary values.

Another important aspect of ambient-awareness is the collection of history information. The provision of ambient-aware services based on actual data is only the first step towards I-centric Services. Based on historical values of ambient information advances reasoning mechanisms can be established. Nearly every activity of an individual user affects the physical environment. Collecting the ambient information caused by a certain individual leads to a historical profile that can be used to predict future activities of that individual. The prediction of future activities can feed the adaptability mechanisms of an I-centric communications system to provide a higher grade of personalization.

3.2.6 Adaptability

An important feature of the Open Profiling concept is adaptability. When an individual uses an I-centric Service, the system has to adapt itself to the individual's preferences and ambient information, which suit to the individual's communication space. The focus of adaptability lies on the user as individual being. The result of an adaptation-process is an I-centric Service for a certain individual.

An individual can adapt a service directly through a Human-Machine-Interface (HMI). For example, the individual can configure the settings of his personal answering machine. This type of customization leads to an immediate adaptation, since it has the characteristic of a direct instruction. The altered settings will be stored in the individual's profile and taken into account the next time a service execution will take place.

In comparison to its direct approach, I-centric Services cause automatic adaptability. A service will adapt itself to the communication space of the individual, which comprise the peculiarities of the individual's current environment. An I-centric Service can for instance adapt itself to the current noise-level of the individual's environment and set the ring-tone volume to an appropriate value. Automatic adaptability is just reacting to certain specified characteristics of an individual's communication space, which includes the individual's current actions and the environmental state.

3.2.6.1 Background

Today’s communications environments are still characterized by the usage of several services. Each of them needs to be accessed in a specific way. The more services a user wants to employ, the more devices or applications he has to operate. There is almost no way to handle a certain telecommunication service with a terminal, which is not designed for it. To over-come this restriction, a new category of telecommunication service solutions has gained momentum in the last years. Under the catchword ‘Unified Messaging’, these solutions have the aim to fulfill the vision of integrating different communication services [Nan97, Gre97].

The usage of different services should be possible in the same way. The idea for such solutions is based on recent developments in the area of personal communications featuring information at any time, at any place, in any form [Eck96]. That means a user should be able to handle several kinds of telecommunication services independent when, where, and on which way he wants to do that. The idea behind these systems is to provide a user with a unified access to all services he has subscribed. Additionally, unified message delivery should be possible. [Arb01c]

Media Gateways – Media Conversion

Facing the functionality of I-centric communications systems, the demand for a variety of conversion technology is obviously. In general, media gateways provide functionality to convert media containers like email, voicemail, facsimile, and short messages into each other. This conversion process is executed on certain converter units. They are capable to convert a specific medium type or format into another one as shown in Figure 35.

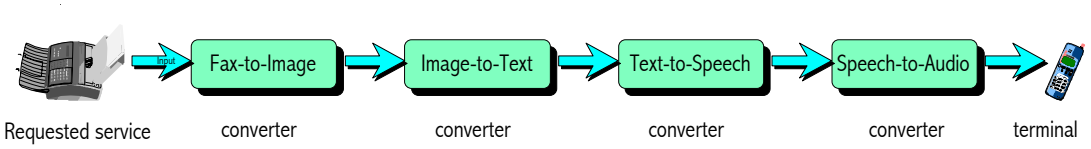


Figure 35: Media Type and Media Format Conversion

For implementing such kinds of media gateways the industry provides 'off-the-shelves' technologies, e.g. automatic speech recognition, optical character recognition, interactive voice response, speech synthesis, text-to-speech conversion, language detection, text filtering, content analysis, etc. These technologies can be combined to a powerful conversion environment, which is able to adapt different media types to each other. [Arb00c, Arb00d]

The idea of media gateways is to establish an intelligent control of these conversion capabilities. A media gateway converts a specific message or stream according to individual preferences and ambient information.

To illustrate the conversion capabilities, which are possible, employing the technologies mentioned above, some examples are given:

Example 1 – deliver an email to a mobile phone: The media gateway has to receive the email, analyze the header, generate appropriate representations of the sender, detect the language the email is written in, pre-process the email for text-to-speech synthesis (by skipping any unspeakable content), apply text-to-speech synthesis to the result, and finally, deliver the produced audio-file via a telephone line to the individual's preferred terminal.

Example 2 – deliver a facsimile to a telephone: The media gateway has to receive the fax, apply a optical character recognition to it, detect the language of the resulting text fragments, analyze included images to ensure their notification, pre-process the text for text-to-speech synthesis (by skipping any unspeakable content), apply text-to-speech synthesis to the result, and finally, deliver the produced audio-file via a telephone line to the individual's preferred terminal.

An important instance has not been mentioned in all examples. Beside the media gateways, another instance is responsible for instructing the media gateways how to adapt the content. In I-centric communications, the I-centric Services have to trigger media gateways to perform any action.

3.2.6.2 Analysis of Incoming Service Requests

The analysis of a service request is a prerequisite to adapt a service in a personalized way. Therefore, two different kinds of analyzing service requests should be taken into account.

Service related analysis

This kind of analysis relates to the special characteristics each service has. It has to be evaluated which persons are the calling party and the called party. Additionally to that, the time and the expected costs will be determined. Service specific implementations have to be employed, to find out this information. For instance, the mapping of telephone numbers to logical representations of individuals will be done by a piece of software, which is able to deal with telephony services.

Content Related Analysis

After finding out the service related information about a communication request, the content, which has to be transmitted, is analyzed. A simple rule can be applied to the content related analysis. The more information the analysis is able to find out, the more flexible the service adaptation can be done. Therefore, each kind of upcoming technologies for content analysis should be taken into account. This comprises technologies like:

- automatic speech recognition,
- optical character recognition,
- language detecting (including automatic spell checking),
- digital image evaluation (including pattern recognition), and
- automatic text search (text robots).

All information, collected by the analysis, will be used to find out how certain content has to be processed. New kinds of technologies for analyzing messages will extend the parameters of service adaptability. A very simple example for applying content analysis is the textual pattern matching in the subject field of an email. If this field contains a specific word (pattern), a predefined action can be executed.

The advanced technologies mentioned above will also help to realize the application based QoS, which has been introduced. Especially, the intelligibility [Pfe99] of a certain message can be analyzed to enable a useful service delivery. In example, a user can have the possibility to block messages, which are arriving in a language, he does not understand. Alternatively, he can force the processing of such message by automated translation processes [W-UNL]. For future telecommunication environments, these kinds of content analysis will produce new value-added services.

3.2.6.3 Rule based Customer Service Control

An individual describes his preferences for different services in rules. An individual can have as many rules as it wants. Each rule consists of two parts, a condition, and a list of actions. The list of actions enables to react to communication requests with several actions. [Arb01c]

Conditions

Conditions define on which circumstances a specific service behavior is demanded by the individual user. The most important conditions are destination related, originator related, time related, service related, contend related, and QoS related. Hence, these five categories should be used to describe conditions generally. Destination and originator related conditions face from which user, service, or terminal or location any information comes in or is to be delivered. Time related conditions describe the time any information comes in or the time any information has to be delivered. These conditions are defined very flexible to comprise periodic events (i.e. send every second day at 6 o'clock a fax, or every second week I would like to have a pizza). With service related conditions, an individual can customize what kind of service he wants to have processed facing its preferences. The content related and QoS related conditions describe all characteristics, which can be determined by the employed content analysis. The more intelligent this analysis is the more parameters can be used for content oriented service adaptation.

Actions

The way, how a communication request has to be handled, is described in actions. Three different kinds of actions can be identified, the destination related actions, the service related actions, the content, and QoS related actions. The destination related actions comprise each kind of forwarding or redirection instruction to other individuals, terminals, services, or locations. Service related actions are capable to force or forbid any kind of service or format. The content and QoS related actions enable even better service adaptation. Different media types for content can be forced as well as forbidden. It is possible to send a fax, written in English, which will be converted to Spain, because Spain was forced for outgoing messages.

Evaluation

If a communication request is signaled to an I-centric communications system, this request will be analyzed. The results of that analysis will be used to evaluate the rules defined by the addressed individual. Rule by rule will be evaluated until one rule is found which matches the requested service. In this context, 'match' means that all parameters, specified in the condition of a certain rule correspond to the service request (e.g., it is the same service, it is the same time, it is the same content, etc.). If a rule matches the request, the action of that rule will be performed. The system tries to deliver the service as specified in an action. As the evaluation is strongly related to the service logic of I-centric Services, this topic will be further discussed in section 3.2.7 where the general model of I-centric Services and the creation of service logic are described.

3.2.6.4 Evaluation of Individual Preferences and Ambient Information

Profile evaluation takes place in each kind of logic that has to be processed for I-centric Services. The complexity of the evaluation results from the interaction and relationship between several profiles.

Example: if an individual user wishes the home automation system to switch off all lights at 10 p.m., because he normally sleeps that time, but today he is watching TV until 11 p.m., the evaluation of his individual preferences and ambient information has to lead to the decision to keep the light on. [Arb00e]

Abstract wishes ('it is too dark here') will be mapped into system specific commands, taking into account the context of that current situation (`'switchON(device_12)'`). The evaluation process has to provide several features for enabling such mappings. It has to be able to map abstract 'wishes' to concrete behavior of certain object. If two or more objects can only realize a 'wish', the interaction between different objects in a user's surrounding has to be ensured by some kind of service logic (e.g. 'keeping the light on while the TV set is running').

The profiling mechanism should also memorize once processed actions for the provision of self-learning capabilities. The more the system can adapt itself to each individual's individual demands, the more an individual will be sufficiently supported. Therefore, each tiny piece of data, which has been collected about a user, has to be stored to be accessible for the future. That opens a huge area of research, which will be influenced by several fields of computer science (human-machine interaction, artificial intelligence, knowledge based systems, and semantic processing).

A profile[24] evaluation mechanism ***A(P)=R*** analyzes or interprets a profile (P) to generate a result (R) applying a specialized algorithm (A). The result of the evaluation defines what has to be done by the system at a certain moment in time.

[24] In general, it is assumed here that individual preferences as well as ambient information are stored in profiles. These profiles are subject to be analyzed by the evaluation mechanism. The evaluation is supported by ontologies applied to the profiles as described in section 3.2.3.2.

Figure 36: Fixed Profile Evaluation

An unsolved problem in this context is the portability or interoperability of algorithms, which have to be applied to profile information. In most existing profiling implementations, the algorithms are 'hard-coded' and therefore not changeable after system startup [IN, W-UMS]. The Open Profiling Framework proposes a flexible profile evaluation mechanism, which gets the actual computing algorithms as parameter like the profile that is to be evaluated. This approach can substitute 'hard-coded' solutions (A) with a generic interpreter (G) for abstract-typed algorithms (A'). The result *G(P,A')=R* provides a flexible solution to overcome the restrictions described above. Figure 36 and Figure 37 illustrate the difference between both approaches.

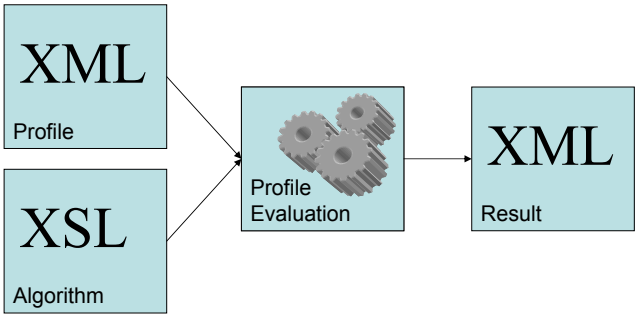


Figure 37: Generic Profile Evaluation

The generic interpreter has two types of inputs (profile and abstract typed algorithm) to generate the same result as the 'hard-coded' solution. Because the actual algorithm is a parameter of the function G, just as profile information, it could be exchanged or manipulated during runtime of the system. The eXtensible Stylesheet Language Transformation (XSLT)[25] is a promising candidate to be used for the definition of the algorithm. An XSL interpreter uses algorithms that have been externally specified in the *Extensible Stylesheet Language* for the processing of XML typed data. Since XSL typed algorithms are compliant to the XML standard, they can be seen as XML typed data and changed through the same XSL interpreter as well as profile information. As this kind of abstract evaluation mechanism has finally to be performed by I-centric Services, more details are given in section 4.4.2 where the realization of the profile evaluation is explained.

3.2.6.5 Learning

Adaptability though learning is the most challenging feature of the Open Profiling Framework. It enables services to adapt itself to an individual's preferences, daily routines, and habits. If an individual checks his stock rates first when it comes to its office in the morning, an appropriate I-centric Service could predict the individual's preferences after three or four repetitions of this daily action. Learning is therefore concerned with observation and training, which makes the history profile of the Context Server valuable for this kind of adaptation. Recent states of an individual's profile can be used for the prediction of actions necessary in the future.

There are two general approaches to adapt system-behavior, which can be described as **'soft-adaptation'** and **'hard-adaptation'**.

In the **'soft-adaptation'** variant, gathered personalization information will be stored in the individual's profile. If an I-centric Service component provides a certain service to this individual, it

[25] eXtensible Stylesheet Language Transformation [W-XSLT]

uses the personalization information inside the individual's profile for the processing of the individual's current context and therefore changes its behavior. This variant is called *soft*, because it just changes profile information and leaves processing algorithms unchanged.

This approach of personalization is used very often today in services provided via WWW. Electronic commerce websites often store profile information about their customers to present news, products, and advertisement to them, which are considered relevant. Such an e-commerce service can for instance process the recent transactions and the considered customer group to predict product and news, which are most likely considered interesting by the user.

By comparison, the **'hard-adaptation'** variant affects the service-behavior more deeply, since it changes the processing algorithms for the respective individual. Hence, an I-centric Service has to be created that processes algorithms, which can be adapted for each user individually.

This adaptation approach marks a new approach towards adaptability, since dynamically and individually changes in the processing algorithms have not been possible in traditional 'hard-coded' service logic approaches. This approach becomes practicable using a Profile Evaluation approach (see section 3.2.6.4), which makes use of a generic interpreter that processes typed profiles with externally specified algorithms. The algorithm in this approach is not part of the interpreter and can therefore be changed for adaptation reasons during the runtime.

The direct adaptation of the Profile Evaluation algorithms offers several advantages for personalization. A 'soft-adaptation' that uses profile information needs a generic algorithm to evaluate every kind of adaptation information. This makes Profile Evaluation complex and slow if a lot of different adaptation information has been collected for a given individual. By comparison, a 'hard-adaptation' will be faster and more open to different kinds of adaptation information at the same time. It is not necessary to provide a more generic algorithm for this approach. On the contrary, such an algorithm is very specialized, since it is only feasible for a single individual, which makes it fast and easy to adapt.

A general problem results from human psychology-aspects on the topic of automatic adaptation and adaptability. Automatic learning-processes for adaptation are cyclic: Individual's behavior is monitored and processed. The system adapts itself to this behavior and the process of monitoring and learning starts again for more sophisticated and refined adaptation. However, humans are hard to predict. They do not want to access their stock-depots and message-boxes always in the same way. Habits and usual behavior are always subject to changes.

Therefore, personalization is not only a question of technological practicability but also of human-behavior. A good compromise has to be found that specifies how far adaptability should be applied to system processes without bothering the individual too much with interaction. A system that permanently changes its behavior according to the individual’s actions conveys the impression of powerlessness and unpredictability to the individual. The individual gets the feeling that it cannot control the system's behavior.

This section gave a first glance on the matter of self-adaptation and learning, since a detailed discussion would probably went beyond the scope of this thesis. It is certainly necessary to introducing experiences of research areas like artificial intelligence, human machine interfaces, and knowledge based systems, to investigate this topic further.

3.2.6.6 I-centric Model for Service Adaptability

Service adaptability is the most complex task of I-centric communications. As discussed in section 3.2, adaptability has to take most of the information available in an I-centric communications system into account. Adaptability takes decisions based on:

- ambient information,
- individual preferences, and
- the context which is or has to be activated.

Service adaptability deals with the evaluation of profiles that store ambient information, preferences, and context definition. The result of the evaluation process determines which context has to be activated or adapted to changed environmental conditions or user wishes.

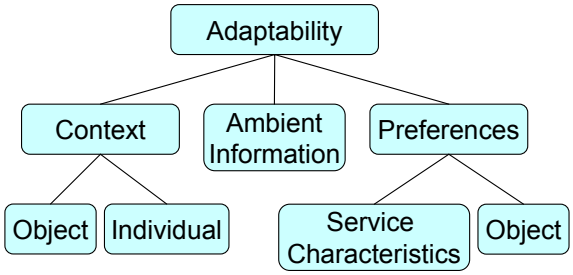


Figure 38: I-centric Model for Service Adaptability

As introduced in the Reference Model for I-centric communications, activating contexts is done by I-centric Services. They contain the service logic that controls service adaptability. For that reason the complete picture, how to adapt I-centric communications systems, is given in the next section, which provides the general model for I-centric Services.

3.2.7 Integrating Personalization, Ambient-Awareness, and Adaptability

I-centric Services represent the central point of service control within an I-centric communications system. They are specified to control certain objects that belong to one or more contexts an individual user has specified. I-centric Services are dedicated to only one individual supporting it in achieving its goals.

As discussed in the last section, I-centric Services activate or adapt contexts based on the evaluation of ambient information and individual preferences. In terms of the Reference Model for I-centric communications this mean, I-centric Services use the information delivered by **personalization** and **ambient-awareness**, providing the control of **adaptability** through their service logic. Figure 39 illustrates this fact and introduces tree new kinds of information, namely triggers, conditions, and actions.

Triggers, **Conditions**, and **Actions** have been introduced so specify the service logic of I-centric Services. Triggers provide filtering mechanism to prevent that I-centric Services have to process any ambient information that is generated. The Trigger defines by receiving what kind of ambient information the I-centric Service must be activated. Conditions define an additional filter that prevents I-centric Services from unnecessary actions. These additional filters must not be focused on the ambient information just received. Moreover, they can be used to check organizational or personal constraints that have to be taken into account for a certain service. Actions define what has to be done by a certain I-centric Service (e.g. activation of a group of objects). [Water00]

For example: A scenario is envisaged where an I-centric Service ‘Constant Brightness’ controls the brightness level within rooms by switching lights according to individual preferences. In this case, the Trigger for the I-centric Service would be ‘I’m entering a room’. The condition would be ‘brightness level in current location ≠ preferred level’. The Action to be taken by the I-centric Services would be ‘adjust the light to my preference’.

If ambient information with the meaning ‘I’m entering a room’ occurs the I-centric Service is activated. When the conditions are true, the actions will be processed. Processing the actions will cause activation or adaptation of objects. Therefore, I-centric Services have to resolve what physical resources are available to fulfill the requests given in the Action. The Open Profiling Framework, which has been introduced in section 3.2, provides the necessary mechanism to gather, exchange, and evaluate the information needed to perform an I-centric Service. The concrete mechanisms how to process I-centric Services are implementation specific. For this reason, these mechanisms are discussed in the realization part of this thesis (see chapter 4).

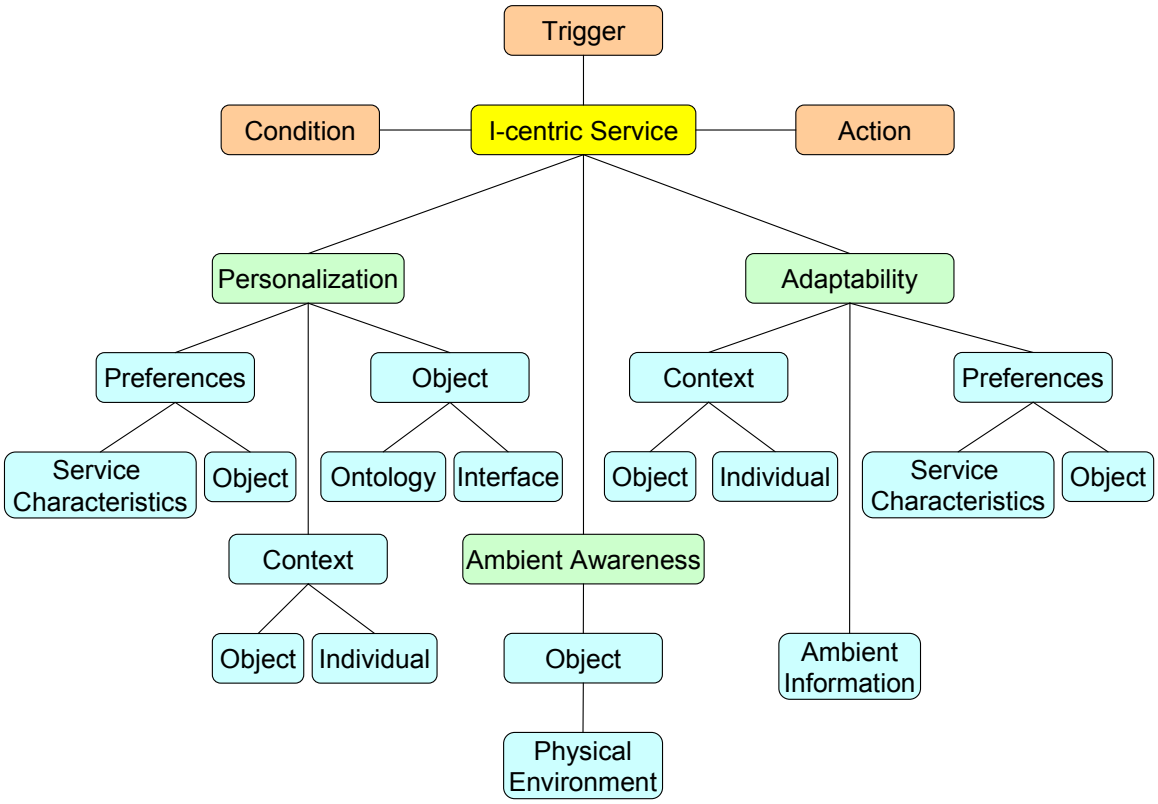


Figure 39: General Model for I-centric Services

Until now, it has been discussed how I-centric Service should operate in general and what features they have to provide (personalization, ambient-awareness, adaptability). As introduced in the vision for I-centric communications, these services should be really personalized and they should perform their tasks for only one individual. As consequence, only the individual itself knows what kind of services it wants to have provided by the I-centric system. The individual has to be provided with tools enabling the generation of new I-centric Services reflecting individual's demands. [Gins95]

Secondly, the frequent behavior of individual users acting in their communication spaces can be monitored and profiled to derive future activities. Such a mechanism can be used to adapt the system behavior by generating new I-centric Services automatically.

In the two following sections, the Interactive Service Creation and the Automatic Service Creation for I-centric communications systems are discussed. **Interactive Service Creation** empowers the individual user to generate his own I-centric Services, whereas the **Automatic Service Creation** creates new I-centric Services based on analysis of recent activities of an individual.

3.2.7.1 Interactive Service Creation

Following the principles outlined in the vision for I-centric communications (section 2.1), individuals are choosing services from objects in their individual communication space. In addition, individuals request a coordinated processing of different services acting in a certain context. Relationships and causalities are assigned to different objects to perform tasks in parallel. In this sense, individuals perform a kind of service composition to achieve their goals or to fulfill their requirements.

Acting in a 'relax-context': An individual user might have a 'relax-context', which describes the relationships and causalities between different objects involved in that very context. As the objects do not relate to any physical resource before being activated, this context can be applied to any situation or location the individual might be in. A typical setting for such a context is therefore something like 'I want to get entertainment news, movies, sport shows, I do not want to be disturbed by any other individual or information, and want to have the room adapted according to my preferences, e.g. quite dark'.

Activating such kind of context means to identify the physical resources available in the current vicinity of that individual and control them according to what is requested by context and the personal preferences. From the perspective of a communication system, a context defines a macro of abstract instructions to be executed in a coordinated manner by the physical environment.

I-centric Services are the instances in I-centric communications systems that activate contexts based on personal preferences and ambient information (see section 2.2.3). The information about contexts, preferences, and ambient information can be requested from the Context Server at the time an I-centric Service is executed. Contexts and preferences have to be specified by individual users to reflect their demands. Ambient information is provided by the physical sensing environment.

The concept of Interactive Creation of I-centric Services follows the idea as described in the following. A user interface should enable individuals to browse their individual communication space, assigning relationships and causalities to existing objects. Additionally, the relationships and causalities should be based on personal preferences and ambient information. The result of the service creation process is a **context**. It can be activated by an I-centric Service under certain circumstances automatically or directly by the individual.

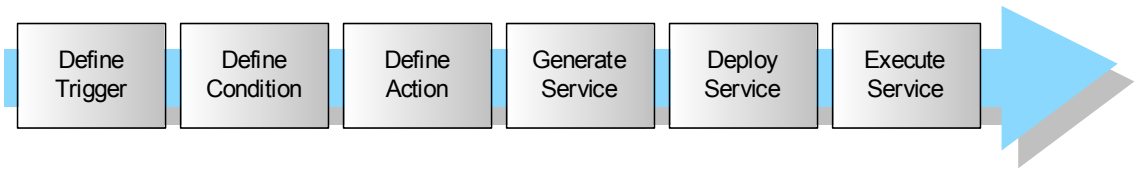


Figure 40: Process of Interactive Service Creation

Based on the general model for I-centric Services (see section 3.2.7), the steps given in Figure 40 are performed during the service creation:

- **define service trigger**: event, time-constraint that requires service execution
- **define condition**: thresholds for ambient information, preferences, or object states
- **define action**: service configuration (operation, parameter) to be performed by objects
- **generate service**: combine service trigger, condition, and action into a context
- **deploy service**: introduce the context to the I-centric communications system
- **execute service**: starts an I-centric Service that activates the generated context on demand

In general, an I-centric Service consists of one or more triggers, conditions, and actions as shown in Figure 41. The trigger defines the situations in which the service has to be activated. From an I-centric perspective, this defines the circumstances that lead to a context activation. Events for triggers can be ambient information issued by objects (including direct commands from an individual), new relations between objects, or time-based constraints. In case, an event, according to a defined trigger, is received by the I-centric communications system, it activates the corresponding I-centric Service.

E.g., a trigger might be a temperature value issued by an object representing a temperature sensor. An I-centric Service configured to control the temperature would be activated when having defined values of temperature as its trigger.

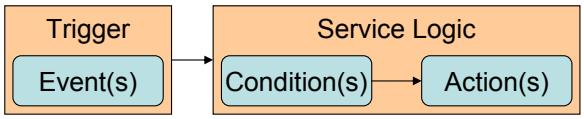


Figure 41: Structure of I-centric Services

After activation, the I-centric Service executes its **service logic**. It considers conditions, personal preferences, and ambient information, which have been selected during the service creation. If the conditions match the current situation of the individual, the actions, which have been also specified during the service creation, are executed. Actions lead to activations of objects in the individual communication space.

During the service generation and context/object activation, the Context Server and the Ontology Server are involved. The Context Server provides the information about instances of objects and relations, whereas the Ontology Server provides general information about available types of objects and relations.

Service trigger

A trigger can be any kind of information that is generated inside the I-centric communications system. Nevertheless, a first structuring of that information has been performed:

- **time-constraint**: defines a time or a time period when the service should be activated
- **sensor-data**: is qualified as any data that is delivered from the physical environment. Such kind of data has to be augmented by Context Interpreters applying domain specific knowledge to it (sensor-data → sensor information)
- **location-information**: is expressed by relations between objects representing locations and other physical entities (even individuals). Typical example: 'Individual at Home'.
- **object-states**: define certain states of objects or other available information that should cause the activation of a context. E.g., 'breaking news should be delivered immediately' or 'TV set is switched to channel 32'.

The combination of such information in one trigger provides a powerful instrument to define when I-centric Services should be activated.

Condition

The same information, as defined for triggers, can be used for conditions. For being evaluated against personal preferences or values specified during the service creation, conditions are defined as **Boolean Expressions**, e.g. 'room temperature ≠ my preferred temperature' or 'time is between 8 a.m. and 6 p.m.'. Such expressions are evaluated by the service logic. If the result is 'true', the actions defined for that service are executed.

Action

Actions define concrete instructions, which have to be performed by one or more objects. These instructions are finally sent to the according object(s). Instructions contain operations and parameters that are compliant to the ontology definition of objects to be controlled.

Actions can be dedicated to abstract objects (without any idea what physical resource is underneath) or to physical objects (to control a certain piece of software or hardware, e.g. to switch on a certain light). The service logic maps abstract objects to physical instances by applying domain specific knowledge to that process.

E.g. to determine what physical terminal has to be selected for information delivery, the location of the individual, the presence of other individuals at this location, available terminals at this location, personal preferences for receiving information, and the content of the information to be delivered has to be taken into account by the service logic. [Arb96] After the service logic has decided which objects have to be controlled how, according instruction are generated and sent to the objects.

Generate service

As discussed in section 3.2.6.4, I-centric Service performs a profile evaluation, which is generic in the sense that the evaluation algorithm is part of the service specification. A generic I-centric Service implementation can be provided that is not specific to any application domain at all. During execution time, the I-centric Service is parameterized with its service logic (the profile evaluation algorithm), ambient information, and personal preferences.

To generate a service finally means to combine the defined service triggers, conditions, and actions to a context that can be activated by an I-centric Service and to generate the service logic that evaluates whether and how the context has to be activated.

Since the description of the service logic is implementation and technology dependant, more details are given in the realization chapter (especially see sections 4.4.2 and 4.4.5). The same applies to the last two steps of the service creation process (deploy service and execute service).

3.2.7.2 Automatic Service Creation

Sections 3.2.7 and 3.2.7.1 have shown a general structure of I-centric Services, their general processing, and an approach to generate and deploy these services in an interactive process. The lessons learned, during the implementation phase of the interactive service generation, resulted in the idea to generate I-centric Services automatically.

To clarify what is meant by automatic service creation of I-centric Services, two short examples are given in the following:

Example 1 – Intelligent facility control: An individual enters a room, switches on the light, stays for a while, and before leaving it switches the light off. The individual repeats this procedure three times. The fourth time it is entering/leaving the room, the light should be switched on and off automatically. The automatic service creation has to recognize this frequent behavior of the individual and has to generate a new I-centric Service. This new service releases the individual from switching the light on/off.

Example 2 – Intelligent call control: During office time, an individual might accept calls from business partners only, and during important meetings, it might accept calls from its boss only. All other calls the individual is blocking immediately when the phone is ringing. The automatic service creation has to detect this behavior and generate a new service, which blocks all unwanted calls automatically.

The basic idea, of the automatic service creation, is to monitor all activities an individual performs in its communication space to derive new I-centric Services from the collected information. The Open Profiling Framework, introduced in 3.2.1, introduces the Context Server that stores any information exchanged inside an I-centric communications system. The Context Server provides a history profile where all data it has ever contained is stored. This profile can be used to restore any historical status the Context Server had. [Gins95, Pent96, Pent99]

As introduced in the last section, the information needed for the creation of an I-centric Service is based on **triggers**, **conditions**, and **actions**. Accordingly, the basic question for automatic service creation is: How to find necessary triggers, conditions, and actions in the history information an I-centric communications system provides.

During the implementation phase of the I-centric communications system, a prototypical implementation has been developed that represents a first step in the direction of automatic service creation. Despite the fact that the automatic service creation was not part of this thesis the encouraging approach selected is explained in the following.

The key technology for automatic service creation is pattern detection. Frequent individual behavior leads to patterns in the historical information. In the first approach, the algorithms that looks for patterns, have been equipped with domain specific knowledge (e.g. facility control

(example 1), or telecommunication (example 2)). These algorithms only look for patterns in historical data that is part of their domain specific knowledge. Consequently, the amount of information to be analyzed is reduced dramatically.

Furthermore, the domain specific knowledge also contains pre-selected patterns (e.g. 'enter a room and control a device'). The task to be performed is to identify which of the pre-selected patterns have been caused by certain individuals frequently. The approach taken so far is quite simple, but has already shown a big potential to be enhanced. Example 1 and 2 can easily be realized be applying this approach.

Next steps should try to analyzed the historical information without pre-selected pattern, and finally without and kind of domain specific knowledge.

3.3 Super Distributed Objects

The reference model for I-centric communications has introduced the concept of objects, which provide services within an individual communication space. This section introduces the concept of Super Distributed Objects (SDO) as enabling concept for I-centric objects.

The basic goal of I-centric objects is to represent a specific hardware or software entity providing well-defined services from the perspective of an individual. Furthermore, an object is responsible for hiding the technical peculiarities of the underlying hard or software. Open Service APIs harmonize the given functionalities. [Stif96] Two different kinds of functionalities have to be taken into account. On the one hand, an object has to provide a well-defined interface for service provisioning (usage), and on the other hand, the same object has to provide and has to use well-defined interfaces to be embedded within a service platform or a respective service execution environment.

The concept of Super Distributed Objects has been developed by the Object Management Group (OMG) Domain Special Interest Group (DSIG) for Super Distributed Objects (OMG DSIG SDO). The work on the reference model for I-centric communications was introduced to OMG DSIG SDO. It has influenced the SDO specification at the OMG. Therefore, the SDO concept follows the ideas as described in the reference model for I-centric communications and extends them to a more technical level. In the following, the concept of Super Distributed Objects is introduced to illustrate how SDOs can be used to setup individual communication spaces.

An SDO is a logical representation of a hardware device or a software component that provides well-known functionality and services. One of the key characteristics in super distribution is to incorporate a massive number of objects, each of which performs its own task autonomously or cooperatively with other objects. Examples of SDOs include abstractions of devices such as mobile phones, PDAs, and home appliances, but are not limited to device abstractions. An SDO may represent a software component and act as a peer in a peer-to-peer networking system. SDOs are organized in an ad hoc manner to provide services in mobile environments. For further characteristics of super distribution, see the OMG SDO Whitepaper [SDO-WP].

An SDO is defined by its properties and the services it offers. An SDO is dynamically characterized by its status. The status of an SDO can be modified through the invocation of its services.

An SDO service is represented by a group of operations that can be executed by the SDO. The service execution can modify the configuration and the status of an SDO. While some SDOs can act fully autonomously, others need to make use of services of other SDOs before they can accomplish own tasks. As an example, a Dimmer SDO with a service providing constant brightness in a room can be considered. It should find an SDO offering a brightness measuring service and SDOs that control light switching devices in the room, to have the possibility to build and execute its own service.

3.3.1 Scope of Super Distributed Objects

The increasing availability of high-performance and low-cost processor technology is enabling computing power to be embedded densely in various devices (e.g., mobile phones, PDAs, and Internet appliances) as well as traditional computers. Furthermore, emerging networking technologies such as wireless LAN, IPv6, and plug-and-play-enabled platforms allow those devices to connect to each other in an easy and ad-hoc manner and to construct a large-scale network of devices that provides various applications.

Much attention is being paid on ubiquitous or pervasive computing driven by these technological advances. A goal of these networking infrastructures is to provide a distributed community of devices and software components that pool their services for solving problems, composing applications, and sharing information. The scope of the SDO specification is the transition and abstraction of those infrastructure technologies that target resource interconnection on highly distributed environments into a higher layer using OMG technologies (e.g. UML and CORBA).

Today, there are several resource interconnection technologies such as Universal Plug and Play [W-UPnP], HAVi [W-HAVi], OSGi [W-OSGi], ECHONET [W-Echonet], and Jini [W-Jini]. They are, however, restricted to specific platforms, network protocols and programming languages, or they focus on limited application domains. No common model-based standards exist to handle various resources in a unified manner independently of underlying technologies and application domains. The objectives of the SDO specification are to abstract the existing resource interconnection technologies into a higher layer and to define their information and computational models in one layer.

3.3.2 Common Features of Super Distributed Object

This section introduces the basic features of SDOs, which have been identified within the requirement analysis for SDO based systems done by the OMG SDO DSIG. In general, this features fit the requirements identified for I-centric objects as described in section 2.1.

3.3.2.1 Clusters of Super Distributed Objects

SDOs can cluster in groups. The group concept can assist to better organization of communication and interaction between SDOs. It is possible to group SDOs with similar properties or capabilities, for example, all SDOs placed in certain location or all SDOs controlling light-switching devices. An SDO can belong to several groups at the same time.

3.3.2.2 Autonomy

SDOs are autonomous entities. Each of them can act fully independent from others. They can interact in a peer-to-peer manner without any dependencies from specialized servers (no client-server paradigm). Whenever two SDOs are connected in a network, they shall be able to use the services provided by the other, without any centralized control.

3.3.2.3 Ad-hoc Communication

One of the SDO characteristics is their potential mobility. It should be taken into consideration that SDOs can appear, disappear, and move along the network rapidly. SDOs have to be able to communicate, to use core functions of the system, and to negotiate service usage. Furthermore, they must provide a description of their services in order to enable an ad-hoc usage.

3.3.2.4 Diversity of Processing Capabilities

There are entities with all conceivable intermediate stages of the possible spectrum of computing capabilities in SDO world, from high-capable computers, Personal Digital Assistants to plain light switches almost without computing power, which are to be controlled. To control all

these heterogeneous entities in a common way, SDOs must provide a well-defined interface (Open Service API).

3.3.2.5 Communication Technology Independence

Every device or software, irrespective of the network communication technology it supports, may have benefits from the SDO approach offering its own services to other SDOs or applications. The prerequisite for participation is its conformance to SDO interface and SDO description standards.

3.3.2.6 Scalability

The SDO cluster can consist of arbitrarily high number of entities and can be very dynamic. Therefore, interfaces and communication protocols must consider this characteristic and include appropriate concepts.

3.3.3 Required Functionality of an SDO based Service Platform

The following sections present the functions that SDO based service platforms should offer to facilitate SDO communication and collaboration. Following scenario can be used as an example of SDO interaction in the system. Several SDOs that control electrical appliances in an office build an SDO environment.

A new SDO joins this environment. The 'newcomer' offers a service that turns off the office appliances (for example, lighting, air conditioning system, etc.) at 7 p.m. The SDOs in the environment should supply some mandatory functions to enable the newcomer to become integrated in the environment so that it would be able to communicate and collaborate with other SDOs. A first collection of these mandatory functions is listed in the following sections.

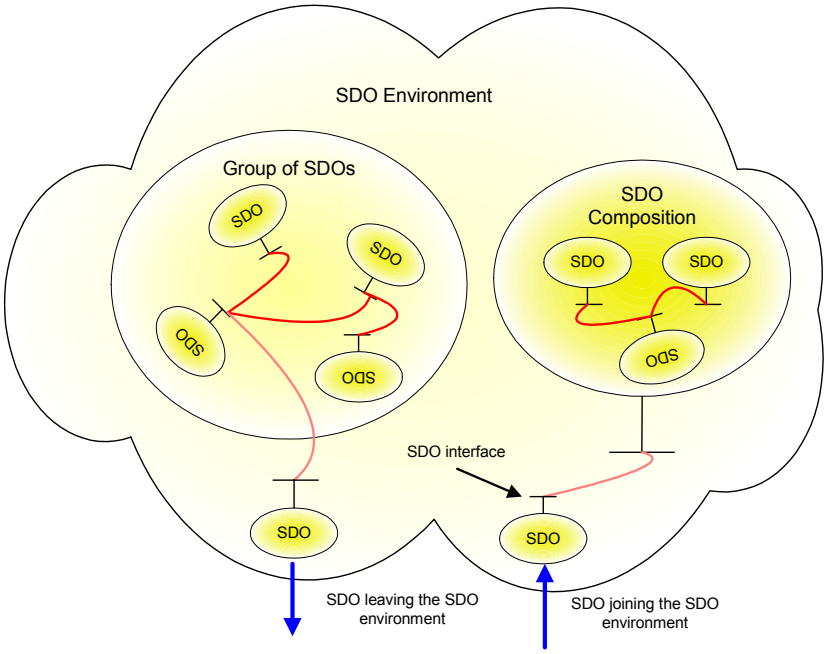


Figure 42: Super Distributed Objects Environment

The scenario given above can be seen as good example of a physical environment realizing an individual communication space. Objects dynamically join and leave the individual communication space due to the movement or changing contexts of individuals. I-centric Services have to discover the objects available in a certain situation to resolve what functionality can be provided based on the available physical resources.

To maintain and to control the physical environment, SDOs have at least to provide the following functions [SDO-RFP]:

3.3.3.1 Discovery

An SDO may need other SDOs to accomplish its services. When joining a new SDO environment, it must discover other SDOs, and their services, required for its own services execution. Knowing what services are needed, the SDO should be able to initiate search for SDOs providing these services. The discovery facility is required by the system to support SDO collaboration. At the same time, a newcomer SDO has to have the possibility to announce offered services to other SDOs or applications in the system.

3.3.3.2 Monitoring

SDO have to provide interface functions to monitor their services' status. A mechanism for the propagation of status information is required. It has to enable SDOs to communicate changes in their status to other system components. Other SDOs or applications can be interested in immediate notification, when the status of an SDO changes. With the introduction of a subscription concept, SDOs may subscribe notification of events they are interested in.

3.3.3.3 Configuration

SDO interface should provide functions enabling configuration of its resource data and service parameters. This function can be used to configure SDO location information, services, alive-notification intervals etc.

3.3.3.4 Reservation

SDOs have to allow reservation of SDOs resource utilization. This requirement is arisen from necessity to get exclusive access to services of I-centric objects.

3.3.3.5 Usage

The usage part provides the service of an SDO to other SDOs or applications. The core functionality of the SDO is offered via an USAGE interface that has to have a well-defined signature. The usage interface has not been standardized by the OMG SDO DSIG. It is common understanding that a usage interface may be a proprietary specification dedicated to a specific application domain. In the context of I-centric communications, the usage interface has to be compliant to ontology descriptions as descried in section 3.2.3.2.

The common functionality, identified in the sections above, can be seen as Generic Service Elements, as requested by the Reference Model for I-centric Communications (see section 2.3.6). They have to be provided by all components of an I-centric communications system to be embedded in the service platform executing I-centric services.

3.3.4 Service Platform Aspects

The Service Platform has to support the development (creation), operation and management of objects and services by means of a execution environment for SDOs, support for deployment (hot-plugging) of SDOs, (re-) binding (configuration) of SDOs, interworking of applications/services and SDOs, and resolving inter and intra SDO dependencies. Therefore, mechanisms and interfaces for the provision of such flexible environments are needed.

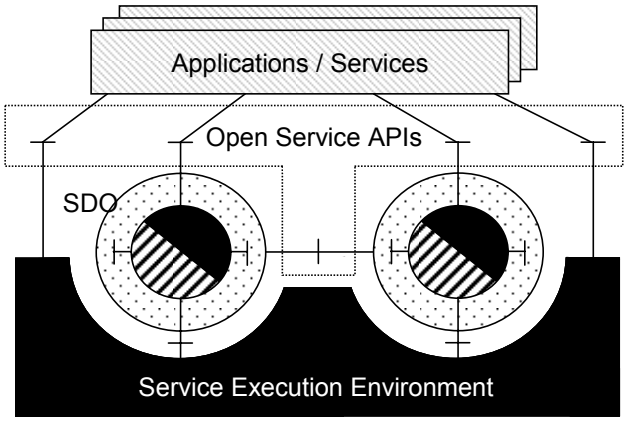


Figure 43: Super Distributed Objects hosted by the Service Execution Environment

SDOs shall be called by development tools and dependencies on programming languages or execution platforms shall be minimized or when possible eliminated. Consequently, SDOs facilitate a much more simplified development of I-centric applications. Corresponding SDO service discovery mechanisms, interworking protocols, invocation methods, APIs, data coding schemas and dynamic configuration and orchestration methodologies are needed.

3.3.4.1 Common SDO Structure for Platform Compliance

An SDO operates as a black box exposing common well-defined interfaces hiding the specifics of all the underlying technologies. An SDO is generic in the way that it:

- can be easily plugged into the service execution environment (service platform),
- can be discovered, monitored, configured, reserved, and controlled using for all SDOs common interfaces,
- can be managed by the service execution environment,
- ensures QoS and security issues by cooperating with the service execution environment
- can communicate with other SDOs,
- can be dynamically grouped with other SDOs for a certain task, and
- it provides all common functions trough well defined Open Service APIs hiding all the underlying technologies that are used to provide a dedicated functionality.

Beside the provision of these common functions, an SDO consists of a specific part that implements the 'real' functionality provided by the SDO towards applications or services. Figure 43 shows how SDOs are embedded in the service execution environment. Services and applications can be build by programming the logic leaving everything that is not directly related to the service logic up to the SDOs by calling functions of the SDO.

Figure 44 illustrates the envisaged structure of an SDO in more detail. Each SDO will provide three different interfaces (**specific interface**, **common interface**, and **common execution environment interface**) compliant to an Open Service API. According to their names, these interfaces provide the functions of the specific part and the common part of an SDO towards services and applications or other SDOs. The **common execution environment interface** established a well-defined interaction point between an SDO and the service execution environment to provide plug'n play and management features. The functions needed to serve this interface are provided also by the common part because each SDO has to support them.

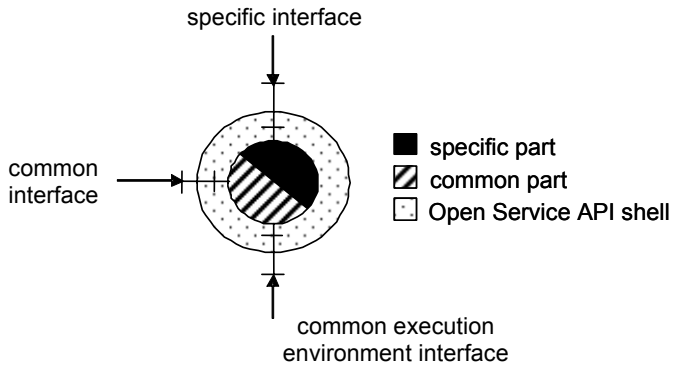


Figure 44: SDO interfaces: Open Service API and internal interfaces

The **Open service API shell** implements the Open Service API towards other components and provides a mapping to the SDO internal interfaces of the specific and common part. That enables more flexibility in developing and implementing SDOs.

3.3.4.2 Benefits of embedding SDOs into a Service Platform

The SDO approach is flexible in terms of reusing once realized SDOs and the fast realization of new SDOs. Application programmers can use well-defined APIs to find, analyze, and use existing SDOs as well as being able to plug new SDOs into the platform. Therefore, a guideline for the development and runtime behavior of SDOs and their interrelations and for the creation of Open Service APIs is needed. Because SDOs can be easily integrated in different kinds of new applications, changing business models can be addressed by just selecting/introducing the right set of SDOs that fulfill the required functionalities. SDO will ease the work of service implementers by providing a set of common functionalities that no longer have to be developed repeatedly.

SDOs will form a highly distributed infrastructure without any inherent need for centralized technical or business control. SDOs cut across the whole service platform and even go beyond that spreading across many physical and logical components like particular wire-line and wireless networks, mobile communication devices, moving vehicles or independent business entities. Autonomous SDOs can group for solving problems, for composing applications, and for sharing contents. Each SDO performs its task either autonomously or cooperatively with other SDOs.

Modifying an SDO does not necessarily involve modifying the application invoking the SDO. The linkage of SDOs is demand driven and dynamic. The SDO composition is a distributed process and independent from compile-time bindings. Sub-sets of SDOs federate to provide functionality that is more complex. That is to reflect changing business roles or to adapt to new system and service demands. Sub-sets of SDOs may form hierarchies, trust and/or business domains. SDOs can act as co-ordination entities for such groups. Replicated instances of SDOs may realize load-balancing networks capable to reflect the geographically distributed mobile networks architectures.

SDOs will enable providers to make their products and services available in a flexible way. The same will enable individual users and applications to discover and use the desired service. Assembly and configuration of contexts composed of various service offers is needed to achieve the requirements stated explicitly or implicitly by individuals.

3.3.5 Common Interfaces of Super Distributed Objects

An SDO is assumed to offer two types of interfaces: operational interfaces (usage) and management interfaces. [SDO-Sub] Through the methods of operational interfaces, the SDO ser-

vices can be invoked. The management interface enables announcement and discovery of SDO services, service control, monitoring, and configuration of its resource data.

Figure 45: SDO Interface Overview

Monitoring, **Configuration**, **Reservation**, and **Discovery** interfaces are shown in Figure 46 belong to the management interfaces, while the operational interface is represented by SDO services (*Service 1*, *Service 2*). The management interfaces comprise common functions (Monitoring, Configuration, Reservation, and Discovery) identified in section 3.3.3, except the usage function. The General SDO Interface comprises basic functions for accessing SDO properties, such as location and SDO type, and for accessing the other SDO interfaces.

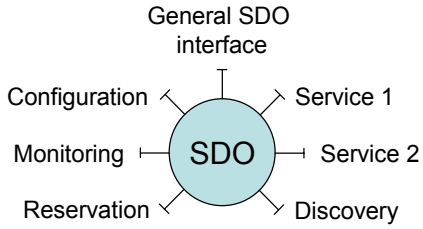


Figure 46: Specified SDO Interfaces

The security issues such as roles defining limitation of access to individual SDO interfaces are not covered in this thesis, but they remain the topic of future activities.

3.3.5.1 Discovery

The functional requirements outlined in section 3.3.3 identified the necessity of a mechanism that allows an SDO to gather information about the cumulative resources and capabilities of all SDOs currently present in the network. There are two complementing implementations of such a discovery mechanism:

- pro-active presence announcements and
- a re-active search.

The pro-active announcement mechanism is a way for an SDO to publicize its capabilities to all other SDOs in the network. These announcement messages provide a description of the services the SDO is providing as well as the status of its presence in the network (whether it is new in the network, been in the network for a while or leaving the network). Announcement messages are spread across the network, possibly by means of multicast or broadcast communication protocols. Furthermore, these announcement messages can be sent intentionally on different occasions.

An SDO can save the announcement message it receives, if the SDO is interested in a service or capability of the announcing SDO. These stored announcement messages can later be used by an SDO to contact directly a proper SDO when needed for a certain service. It keeps the references to both of these SDOs.

However, the drawback of using such announcement messages alone is that new SDOs have to wait for such announcements before they can try to contact other SDOs. Additionally, SDO might accumulate stale announcement data if they miss messages due to network problems or processing congestions.

To bypass these drawbacks and in order to improve the interconnection abilities, an additional search mechanism is provided. The search procedure enables SDOs to find specific SDOs or services in the network.

SDO Announcements

The announcement mechanism gives SDOs the possibility to spread information about them across the SDO network. To participate in the announcement processes, an SDO must possess the capabilities to create and publish announcements, to receive and interpret somebody else's announcements and to store them if needed. On leaving, the SDO should inform other participants about it, so that they can update their information about available SDOs and services.

An SDO can send three different types of announcement message to the network:

- NEW : the SDO is new in the network.
- ALIVE : the SDO still exists in the network.
- LEAVE : the SDO is leaving the network.

Arrival: When joining a network, an SDO sends an announcement message across the network to notify network participants about its existence and to provide information about itself. The other SDOs receiving the announcements will interpret it in order to learn if the advertising SDO is useful for their own services (e.g. if the SDO provides services that the receiving SDO might need now or later). In this case, the SDO can store the announcement (or the reference to the SDO) and refer to the sending SDO. Since, the announcement message includes the address of publishing SDO direct contacts between SDOs are possible without the need for a mediator.

Leaving: When leaving the network, the SDO sends announcement message *LEAVE*. Upon receiving such messages, other SDOs will react accordingly. For example, if *TemperatureController* SDO receives a message specifying that the *Airconditioner* SDO has left the network it will remove its reference to the *Airconditioner* SDO and will not try to invoke its service.

However, since SDOs can leave the network unexpectedly (in case of network failure), without being able to send the LEAVE messages, a mechanism is needed to ensure that other SDOs can determine that the SDOs that left the system is no longer existent in the system. For this purpose, the announcements include the period specifying the validity of such messages. Before the validity of such messages expires, the SDO must renew it by sending ALIVE message. If an announcement from an SDO is not received before the validity expires, other SDOs should presume that this SDO no longer exist in the system.

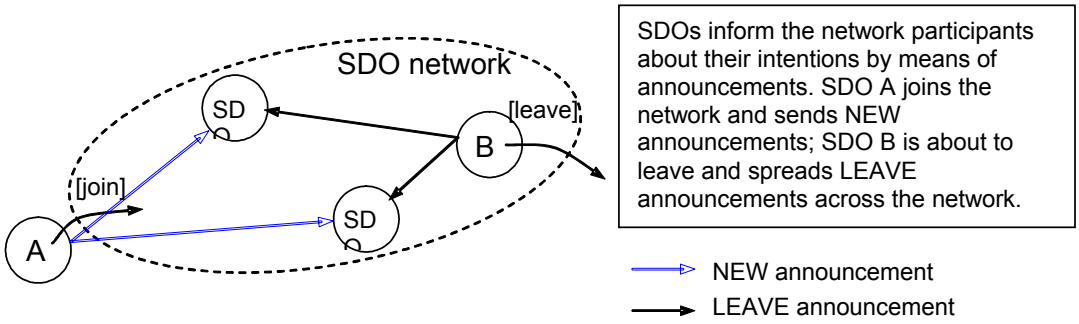


Figure 47: SDO announcements.

The process of SDO announcements is shown in message sequence chart in Figure 48. An SDO that joins the network sends the *NEW* announcement to all SDOs in the system (messages 1 and 2). The announcement message contains the period of the announcement validity. If this period expires and the SDO remains in the system, it broadcasts *ALIVE* announcements (messages 3, 4). The SDO broadcasts *ALIVE* announcements during its presence in the network shortly before the previous announcements become invalid. When leaving the network, the SDO sends *LEAVE* announcements, so that the SDOs that have cached its announcements can remove them and free their resources.

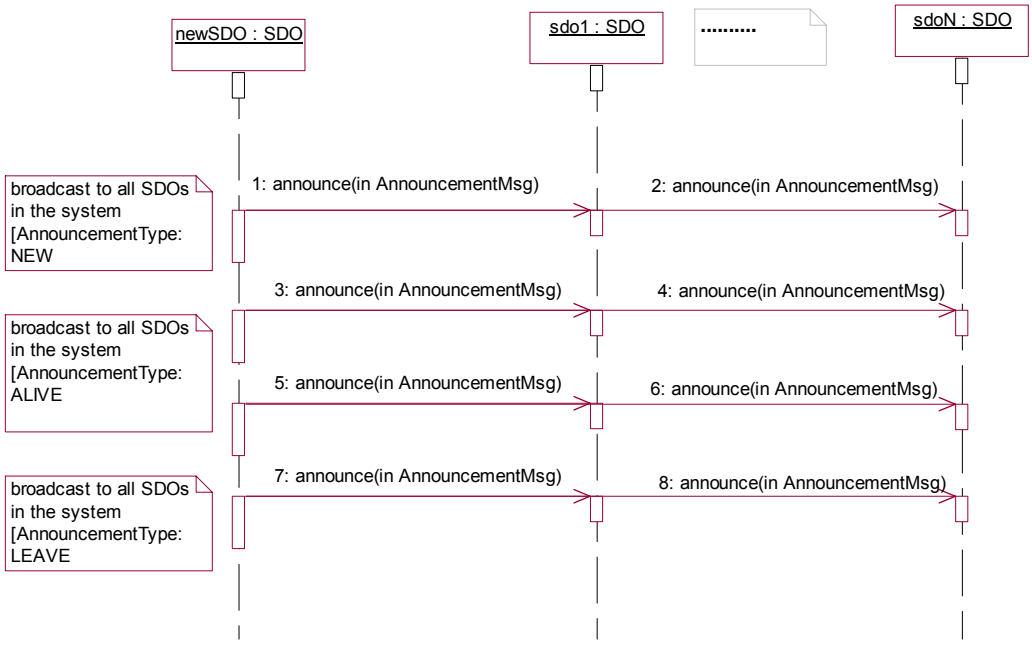


Figure 48: Message Sequence Chart: Announcement

Search

SDOs can use the *Discovery* interface to search for SDOs or services they need. For search purposes, data structures for request specification are defined. For the definition of search criteria, search masks are used. The appropriate properties of the requested SDO are specified in *SDOMask*; device properties can be defined in the *DeviceProfile* mask, and service properties in *ServiceMask*.

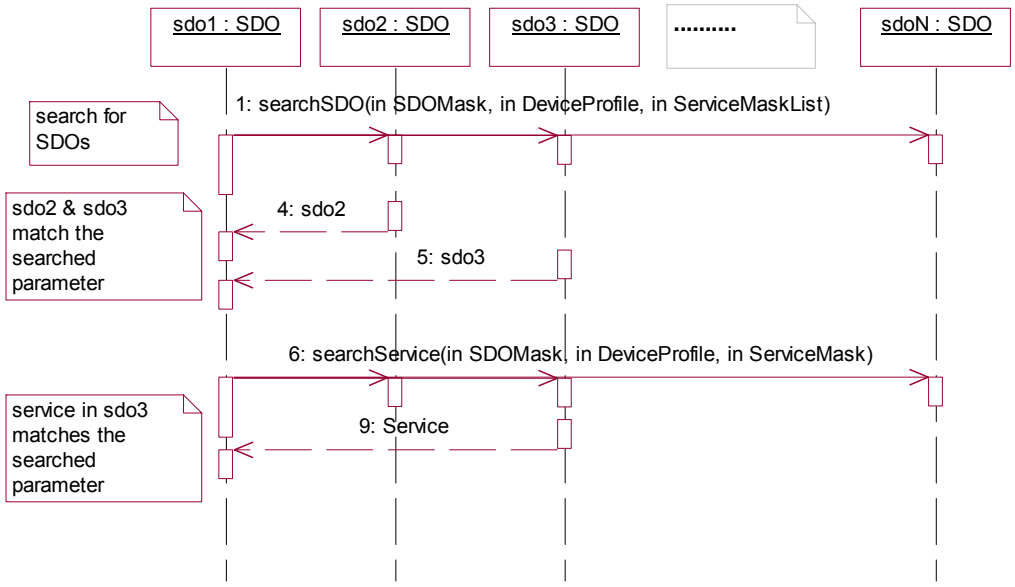


Figure 49: Message Sequence Chart: Search

The search process for SDOs and services in network is shown in Figure 49. The searching SDO (*sdo1*) sends to all SDOs in network the search request with search conditions specified in the appropriate masks (message 1). The SDOs whose properties fulfill the search criteria (*sdo2* and *sdo3* in the diagram) answer the request by sending their addresses back to the searcher.

The format of the addresses depends on the SDO basis technology. For example, in CORBA-based SDO middleware the SDOs respond with their object references. Subsequently sdo1 can use these addresses to access SDOs and request details of their properties or services.

The procedure of search for services is similar to the search for SDOs. The searching SDO spreads the request with the appropriate search masks across the network. The SDOs that provides services matching the criteria can respond by sending back the addresses at which these services can be reached (in the diagram, *sdo3* sends back the address of its service). As outlined before, the format of these addresses depends on the basis technology of SDOs.

The SDO announcement and search mechanisms described above are assumed to operate in a decentralized system. In this case, when publishing services by an announcement or searching for other SDOs' services, the announcement/search messages must be broadcasted to all SDOs present in the network.

Unlike this, in systems with a centralized discovery service, all the data concerning the SDOs currently available in the system is collectively stored in the server. Consequently, there is no need to broadcast announcements to all SDOs - it is sufficient to send the announcement messages only to this central entity. The search for an SDO or service can be performed in the similar way, by sending a search request to the discovery service. Besides, since the discovery service possesses information on all SDOs available in the network or group, it is possible to retrieve information about all SDOs as a group.

3.3.5.2 Monitoring

The status of an SDO can be observed by other SDOs. The SDOs watching the status of the SDO are called *observers* further in this section.

The Monitoring interface provides a mechanism to monitor the various properties or parameters of an SDO. Each SDO implementing the Monitoring interface must specify the parameters that can be monitored. For example, a Thermometer SDO has provided its property *value*, which mirrors the current temperature of the room, as a monitoring parameter. All other SDOs are able to monitor this parameter and if required react on any changes. For example, the Airconditioner SDO turns on if the temperature of the room (from a Thermometer SDO) goes below the predefined level.

To monitor a parameter in a foreign SDO the monitoring SDO must first subscribe to that parameter. The Monitoring interface provides general operations to get information about the parameters that another SDO offers to monitor as well as operations to subscribe to these parameters.

However, the Monitoring interface is optional, that is not every SDO may provide this interface. If an SDO provides the Monitoring interface then it should keep track of all subscriptions of its parameters in order to notify using the given callback interface. For example, the Thermometer SDO must keep track of the Airconditioner SDO to send the notification to it when the temperature is changed.

The Monitoring interface permits to observe the SDO status actively by polling. Furthermore, it offers the possibility to be notified by the SDO when its status changes. Either the whole status variable set or any desired of its subsets can be monitored.

The monitoring by polling consists in periodical checking the status of the SDO by the observer. The process is performed as shown in Figure 50.

If an observer would like to trace the state of certain SDO, at first it should retrieve information about the parameters that specify it. It can be achieved by invocation of the Monitoring interface operation `getMonitoringParameters()`. The ParameterList object returned as the result contains list of names of status parameters and their respective types.

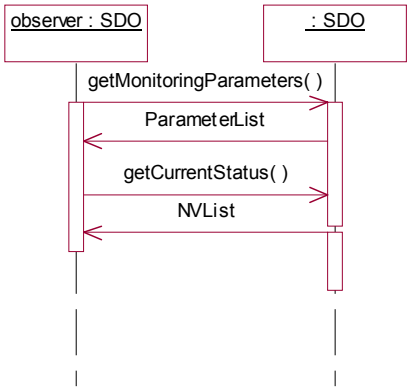


Figure 50: Monitoring by Polling

To get the current state of the SDO, the observer has to invoke the operation `getCurrentStatus()`, which returns the list of parameter names and their values as a name value list. Knowing the respective types of status parameters, their values can be interpreted properly.

If the observer is interested in the current value of some particular status parameter, it can inquire information concerning solely the parameter of interest. This can be done by the invocation of `getParameterValue()` operation with the string name of the parameter as argument. The returned value should be interpreted accordingly to the parameter type.

The monitoring of service state by polling mechanism can be disadvantageous, since it is difficult to estimate the appropriate polling frequency. If too high frequency is chosen, it can lead to unnecessary network workload; if it were too low, the service client would get the information about the new service state long time after it has changed. For this reason, event notifications can be used as the complement to the polling operations.

Event notifications are short messages containing information about current SDO status; they are sent across the network to inform other objects about SDO status changes. With such notification subscription mechanism, SDOs that would like to watch the status of certain SDO (called further *notification subscribers*, or *subscribers* for short) subscribe to status notifications of relevant SDOs. Event notifications are sent to all subscribers when the SDO status changes.

There are several possibilities to predefine when event notifications are sent to subscribers. The subscribers can be informed on the status change immediately as soon as it occurs. This notification option can be called *on-change notification.* This option is useful if the subscribing SDO wants to be notified as soon as one of the subscribed parameter value has changed. For example if the Airconditioner SDO subscribes temperature value in the Thermometer SDO with *on-change* mode, it receives notification about the changes in the temperature value every time the temperature value changes in the room.

The second option is to give the subscribers the possibility to predefine how often they would like to get information about the SDO status. If the subscriber chooses this option, the notifications are sent to it once in specified time interval, no matter if the status has changed since last notification time.

This option is called further *on-interval notification* and is useful if the subscribing SDO wants it to be notified periodically, because it may want to observe the development of a specific parameter over a longer period. In the example of the Airconditioner SDO, subscribing temperature value will receive the temperature of the room only in the specified time interval. Subscribing with this mode may have advantage against subscribing *on-change* if the thermometer is very sensitive and senses very little variations of the temperature in the room.

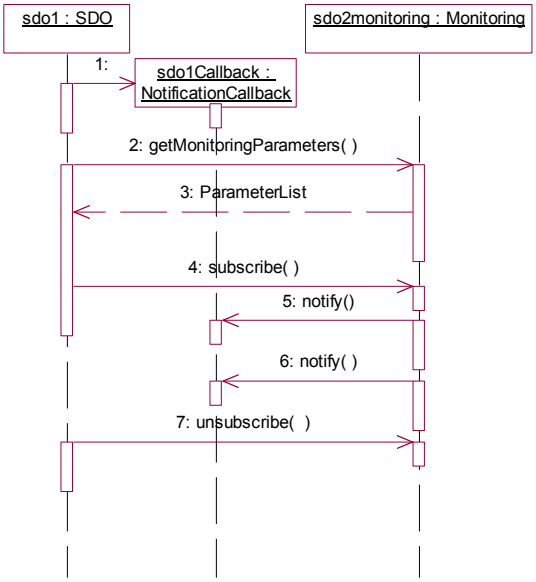


Figure 51: Monitoring by Notification

An SDO can subscribe event notifications by invocation of the subscription operation `subscribe()`. As operation argument, the structure of type NotificationSubscription is required. The structure comprises following data:

- The list of names of status parameters on whose status the subscriber would like to be informed, the callback address of subscriber; the entity whose address is specified here have to dispose of the operations of NotificationCallback interface;
- The notification mode (*on-change* or *on-interval*);
- The time when the subscription should start
- The duration time of the subscription;
- For the subscription of interval notification, the notification interval should be specified additionally.

In general, any properties that can change its value after certain period can be considered as monitoring properties. The temperature value of the Thermometer SDO is a good example of a monitoring parameter. However, an SDO can decide all or any other properties as monitoring parameters as well. For example, Thermometer SDO can decide its service properties *minTemp* and *maxTemp* to be monitored, so that other SDOs, interested in that value, are notified.

3.3.5.3 Configuration

Each SDO or service can possess a unique set of configuration properties, which can be read and modified through the Configuration Interface. SDOs and SDO services that want to provide configuration operations for its properties it must implement the Configuration interface.

The Configuration interface offers the possibility to configure certain properties of an SDO. The implementers of each SDO can decide which object properties will be configurable through the Configuration interface. Since both the SDO class and the Service class implement this interface, an SDO and its services can be configured independently. The configuration process can go as it is shown in Figure 52.

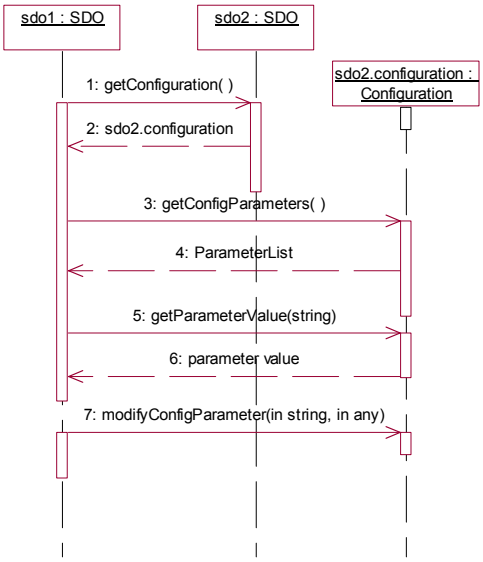


Figure 52: Sequence Chart: Configuration

In the first step, the SDO (or administration application) that is going to configure another SDO have to know the configuration parameters. For this purpose, the Configuration interface operation `getConfigParameters()` should be invoked.

The operation result is a structure of type ParameterList that contains information about parameters, namely their names and types. To get the current configuration of the SDO, the operation `getConfiguration()` can be invoked. It returns the list of SDO configuration parameters with their present values. The `modifyConfigParameter()` operation sets the specific configuration parameter to the given value. The configuration parameter value may then influence the SDO behavior and its services execution.

For example, the service of the Airconditioner SDO could provide its parameters *minTemp* and *maxTemp* as configuration properties. Hence, this service must provide configuration interface.

3.3.5.4 Reservation

The reservation of an SDO service ensures that the reserving object can invoke the service operations at any time without any hindrances from other SDOs. Each SDO services must be reserved individually. A reservation message is sent to an SDO to reserve a service.

This message contains among other data the desired period of reservation to avoid situations when a service remains reserved for indefinitely long time because the object that had reserved it left the network suddenly, without canceling the reservation. Since the implementation of this interface is optional, an SDO does not have to provide it. If it does not provide this interface then any of its services can be invoked anytime without any restrictions.

The implementation of the *Reservation* interface is optional. The *Thermometer* SDO does not provide any service so it may not provide this interface. Though the *Airconditioner* SDO may not provide the *Reservation* interface, it is better for the *TemperatureController* SDO if it is provided. For example, when the temperature of the room drops below the specified level it will reserve the service to turn the heater *on* and invoke it.

When the temperature reaches the desired value then it can remove the reservation and turn the heater *off*. During this period, other SDO are not able to turn the heater *off* or change is configuration parameters (*power*). In case no *Reservation* interface is provided and if an SDO during this period turns the heater *off* or turns the cooler *on* then the *TemperaturController* SDO will keep on turning the heater *on* because the temperature is still bellow the desired level.

Reservation Modes

There are two modes of reservation defined:

- *Exclusive* mode of reservation means that for the reservation period, the reserved service is provided exclusively to the reserving SDO, and no other SDO in the network can either reserve or execute the service. The other SDOs cannot modify its configuration parameters as well.
- *Non-exclusive* reservation mode means that the reserved service can be used and reserved in the non-exclusive mode by other SDOs in the network. Therefore, a service can be reserved non-exclusively by several SDOs at the same time. However, an exclusive reservation is not accepted. Note: While being reserved in no-exclusive mode, a service can be accessed both by SDOs that have non-exclusive access rights and by SDOs that have made no reservation before.

Accordingly, the reservation status of a service at every point in time is free, reserved, or exclusively reserved.

SDOs can retrieve the reservation status of a service by making a status request. The response contains the current reservation status (free, reserved, or exclusively reserved). If the service is currently reserved the remaining duration of reservation is also specified in the response.

3.3.6 Sensing and Controlling Environment

As introduced in section 3.2.1, the Sensing and Controlling Environment represents the physical environment augmented by various kinds of sensor technologies, heterogonous network infrastructures, and terminals that provide ambient information for I-centric Services.

The Sensing and Controlling Environment gathers ambient information about physical facilities or individuals (e.g. detecting the temperature in a room or a command spoken by an individual). The gathering and exchange of ambient information, within the physical environment, is characterized by proprietary formats and protocols. SDOs have to be established on top of the Sensing and Controlling Environment that hide the heterogeneity of the underlying infrastructure.

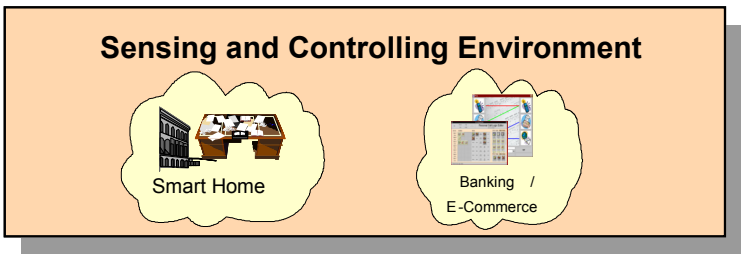


Figure 53: The Sensing and Controlling Environment

The ‘SDO specific part’ (see section 3.3.4) implements the handling of the proprietary protocol or formats. Furthermore, SDOs map the proprietary formats and protocols to well-defined signatures, registered as ontologies within the Open Profiling Framework.

3.3.7 Individual Communication Space based on Super Distributed Objects

Figure 54 illustrates the human communication space, with the SDO interface added to all the objects. This follows the basic idea as envisaged by this thesis. All objects that should be controlled by an I-centric communications system have to provide interfaces compliant to the SDO specification. By providing theses interfaces, objects fulfill the requirements to be controlled and managed by I-centric Services and the I-centric communications system. The SDO interfaces are used to discover the provided functionality dynamically. Furthermore, the SDO interfaces are used to control and to manage the objects according to system’s requirements and individual’s demands.

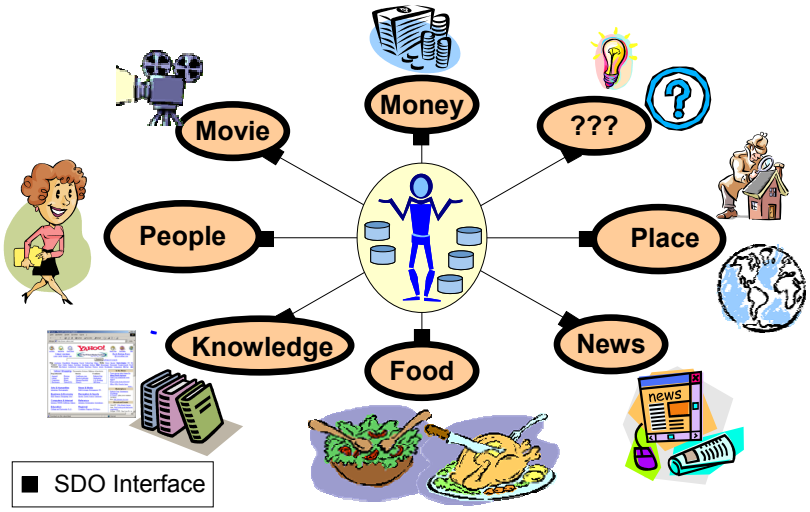


Figure 54: I-centric communication space based on SDO infrastructure

The functionality provided by SDOs, to group dynamically, supports the concept of human communication spaces directly. The human communication space is growing and shrinking over time. This fact can be simply realized by SDOs joining or leaving an SDO community.

3.4 Summary

This chapter introduced an architectural framework following the Reference Model for I-centric Communications. The architectural framework consists of two parts, namely the Open Profiling Framework and the concept for Super Distributed Objects. The Open Profiling Framework describes the modeling, creation, management, and execution of I-centric services. Super Distributed Objects represent the physical resources of individual communication spaces, which are to be controlled by I-centric services.

4 Realization

This chapter introduces the implementation, which is based on the architectural framework introduced in chapter 3. The modeling and the implementation of the Open Profiling Framework and the Super Distributed Objects are discussed in detail. Starting with a complete description of all implemented components, this chapter investigates furthermore the runtime behavior of the realized system, the necessary maintenance tools, the developed I-centric Services, and SDOs.

4.1 Introduction

The vision for I-centric communications follows a top-down approach by analyzing what is necessary for personalized, ambient-aware, and adaptive services. Additionally, one basic assumption is that there is no need for new technologies to realize I-centric Services. The implemented I-centric communications system, which is introduced in this chapter, has been realized based on off-the-shelf middleware and standardized Internet/Web technologies.

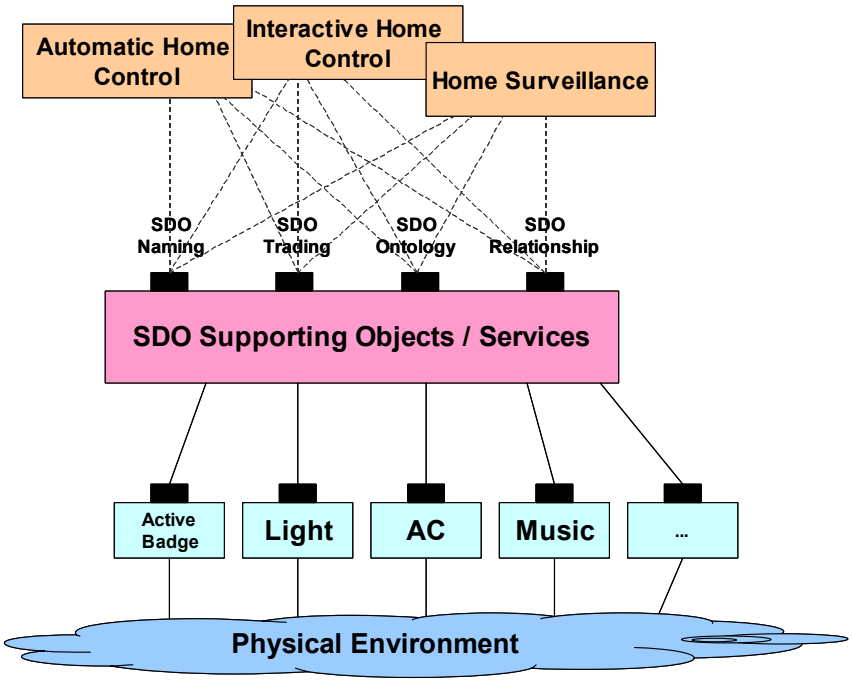


Figure 55: Realized Scenarios

Since the work on the I-centric communications system was embedded in different projects, the software development and runtime environment was selected to ensure flexibility, openness, and interoperability between several project partners. To preview what is later on described in detail, the implemented I-centric communications system is completely based on Java™ technology using open source libraries for XML/XSL processing. The interworking between the distributed system components is realized by using CORBA[26] communication.

The developed components of the Open Profiling Framework (Context Server, Ontology Server, I-centric Service, Context Interpreter, and Service Builder) and the Super Distributed Objects,

[26] The Common Object Request Broker Architecture (CORBA) allows distributed applications to interoperate (application-to-application communication), regardless of what language they are written in or where these applications reside. The CORBA specification was adopted by the Object Management Group to address the complexity and high cost of developing distributed object applications. CORBA uses an object-oriented approach for creating software components that can be reused and shared between applications. [W-CORBA]

as enabling communication environment for I-centric communications, are introduced in the following.

Finally, this chapter gives an overview about the realized scenarios, I-centric services, and SDOs. This is done by showing some scenarios as depicted in Figure 55, which have been developed to demonstrate the potential of the I-centric communications system.

4.2 Context Server

The Context Server manages ambient information and individual preferences generated by Super Distributed Objects, Context Interpreters, and I-centric Services. The following sections describe:

- the Context Server interface,
- the way how contexts, relations and history data is handled by the Context Server,
- the resource data model of the information handled by the Context Server, and
- the Context Server Maintenance Tools.

All information handled by the Context Server falls into two categories: **SDO data** and **Relation data**. SDO data contains ambient information, individual preferences, and information about SDO instances. Relation data contains relation between SDO data. Context Server stores these two types of data separately as *Context* and *Relation Profile*.

The Context Server is also responsible for managing historic information including changes over time (*History Profile*). Like context information, this history profile is also separated in *SDO* and *relation* data.

Any incoming information is structured based on categories provided by the Ontology Server. The information is stored together with its categorization in the *Context Profile*. The semantic information provided for categorization can be seen as metadata about the actual incoming information. Relationships between different entries inside the Context Profile are modeled and stored in the *Relation Profile*.

Example: A relationship describes the fact that a certain SDO (e.g. Light) is located in a certain room of a building. SDO (light) and room are represented in the Context Profile. Therefore, the Relation Profile contains a typed relation (‘is located at’) between these parts of the Context Profile. The type of the relation is part of the relation’s ontology.

The actual syntax and semantic of the information representation inside the profiles depends on the application domain of the respective SDO, Context Interpreter, or I-centric Service that sent it and is not necessarily known to the Context Server. The responsibility of the Context Server just comprises storage and management of profile information, which does not imply knowledge about the semantics of the information.

Both, the Context Profile and the Relation Profile are directly accessible, i.e. other components can add, modify, or delete data. The History Profile has a slightly different role, since it contains a complete history of all changes in the two other profiles. It is automatically generated and provides only reading operations for accessing history data. With this restriction, modifications on the history data are withhold.

4.2.1 Context Server Interface

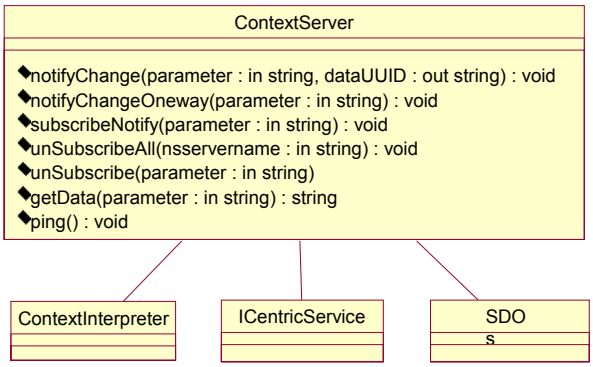


Figure 56: Context Server Interface

The Context Server interface (see Figure 56) offers two different kinds of operations: management operations for administrative purposes and usage operations for accessing the Context Server functionality. The parameter value sent to the Context Server for these operations are of the type string containing an XML document compliant to the Context Server XML Schema specification.

The usage part of the interface has operations for data retrieval and data modification. Requested data sets are retrieved with a request specific search string. This search string queries for information, which is stored in the different profiles the Context Server manages. Modification of information can be done in context profile and in the relation profile. The Management part of the interface contains operations for subscription purposes.

Operations for usage	
Operation	Description
`getData`	Retrieve data from one of the profiles (Context, Relation, or History Profile).
`notifyChange`	Insert, modify, or delete data for a given categorization in a profile. Returns the unique ID (UUID) of the to be notified data.
`notifyChangeOneway`	Does the same thing as the operation above, but does not return unique ID.
Operations for management	
`subscribeNotify`	Subscribe to changes in profile information of a given categorization. Changes will be reported to a Context Interpreter interface.
`unSubscribeAll`	Removes all previous subscriptions.
`unsubscribe`	Removes a specified subscription.
`ping`	Verify if Context Server is up and running

Table 1: Operations of the Context Server interface

4.2.1.1 GetData Operation

There are two different ways to retrieve data from a specific part of a profile. First, a component can obtain the data by using the data retrieval operation (`getData`), or it can subscribe to changes of the corresponding part of the profile, and it is automatically notified about changes. The Context Server provides an operation that adds a subscription for a specified set of categories as well as an operation that deletes all previous subscriptions or a specified subscription.

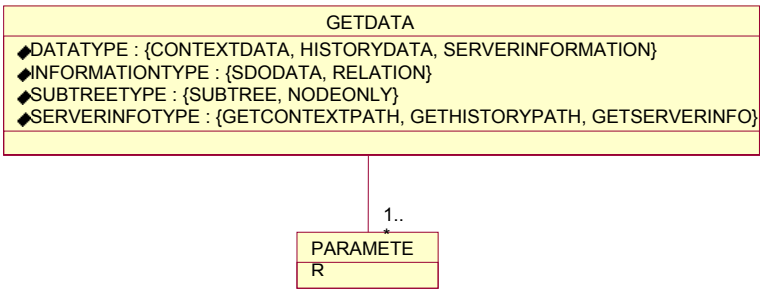


Figure 57: Parameter of the getData operation

The `getData` operation is used to retrieve data from a specific profile, which can be the *Context Profile*, the *Relation Profile*, or the *History Profile*. Parameter defining the `getData` operation is the String value of the XML data. The `DATATYPE` attribute of the `GETDATA` element defines the type of data requested. It is set to `CONTEXTDATA` to retrieve data from the Context or Relation Profile. It is set to `HISTORYDATA` to retrieve data from the History Profile. `INFORMATIONTYPE` defines whether the data is from SDO or RELATION profile. A detailed description of the data to be retrieved is given in the `PARAMETER` element in the XPATH format. The XML Path Language [XPATH] is used by XSLT [W-XSLT] as a sub language for addressing parts of an XML document. For further details, see section 7.1.9.

```
<?xml version="1.0" encoding="UTF-8"?>
<GETDATA DATATYPE="CONTEXTDATA"
         INFORMATIONTYPE="SDODATA"
         SUBTREETYPE="NODEONLY">
  <PARAMETER>
  /DATA/SUBCONTEXT[@TYPE='STATIC'][(SUBCONTEXTELEMENT/@NAME='LOCATION' and
                  SUBCONTEXTELEMENT/SUBCONTEXTELEMENT/@NAME='ROOM')]
  </PARAMETER>
</GETDATA>
```

Above XML document is an example of a parameter to get data from the Context Server. Above statement gets context data (`DATATYPE="CONTEXTDATA"`) from *Context Profile* (`INFORMATIONTYPE="SDODATA"`) corresponding to the XPATH statement defined in the element `PARAMETER`.

To retrieve data from the History Profile, additional information can be added to specify the requested data. Since the History Profile stores all changes over the time, information about the period of the requested data can be added to narrow the search.

4.2.1.2 NotifyChange Operation

All SDOs, Context Interpreters, or I-centric Services, which have generated new data, send the data to the Context Server using the `notifyChange` or `notifyChangeOneway` operation. The Context Server stores the notification in the appropriate *profile* with a unique ID. In `notifyChange` operation, this unique ID is returned to the sender. The parameter passed has the structure shown in Figure 58.

In the following, the elements of the parameter are described in detail.

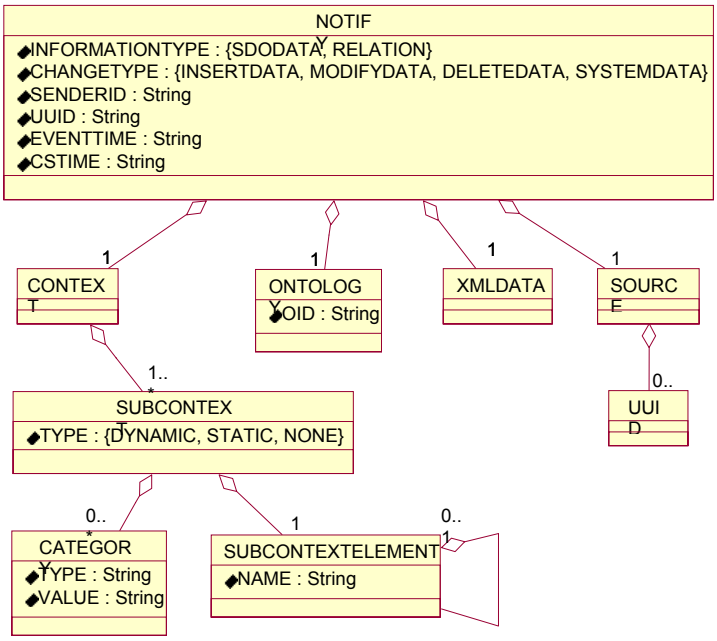


Figure 58: Parameter of the `notifyChange` and `notifyChangeOneway` operations

Element NOTIFY

The topmost XML element for this parameter is `NOTIFY`. This `NOTIFY` element contains information about the context of the data, ontology information of the sender, and the data itself, represented by the elements `CONTEXT`, `ONTOLOGY`, and `XMLDATA` correspondingly. It can also contain element `SOURCE` describing the source object or component of the included data.

Attributes	Description	Value
`INFORMATIONTYPE`	Type of information: SDO or a relation data	`'SDODATA', 'RELATION'`
`CHANGETYPE`	Intended action of the data. - INSERTDATA: New data - MODIFYDATA: Modification to existing data. - DELETEDATA: Delete the data from the Context Server - SYSTEMDATA: system data - REGISTERDATA: Contains registration data of a sending component - DEREGISTERDATA: contains deregistration data of a sending component	`'INSERTDATA', 'MODIFYDATA', 'DELETEDATA', 'SYSTEMDATA', 'REGISTERDATA', 'DEREGISTERDATA'`
`SENDERID`	Identifier of the sender.	any string
`UUID`	Unique ID of the context data. When the Context Server gets a `notifyChange` message, it stores the data with a unique ID. If UUID value is already set in the message, the Context Server saves the data with this unique ID.	unique stringified ID
`EVENTTIME`	Time when the event took place.	string defining date and time in XML Schema date time format
`CSTIME`	Time when the Context Server received the message.	string defining date and time in XML Schema date time format

Table 2: Attribute Description of the NOTIFY element

Element CONTEXT

The **context**, described in this element, categorizes the data in a meaningful structure, acting as identification of the data. By the use of this identification, any components can have access to this data. The **context** can contain one or more `SUBCONTEXTs`. Please refer to section 4.2.2 and section 4.2.2.2 for a detailed description of the categorization of the context data.

Element ONTOLOGY

Ontology element defines the ontology of the data. In this element stores only the unique identifier of the assigned ontology, which is stored in the Ontology Server.

Element XMLDATA

Since the structure of the XML data is not fixed, it cannot be defined inside the Context Server XML Schema. To insert the data in the parameter it is necessary to enclose it in an XML CDATA section. Every kind of XML data can be stored inside this section, since it is not validated against the schema. The CDATA section looks like this:

```
<XMLDATA><![CDATA[  ....xmldata....  ]]></XMLDATA>
```

Element SOURCE[27]

If the notify data is for relation data, it can also have a `SOURCE` element identifying the source of the data. The `SOURCE` element contains the unique IDs of the SDO data comprising the relation specified in the data. For example if an `ACTOR` is detected in a room, the notification '`ACTOR in LOCATION`' is sent to the Context Server. Accordingly, the `SOURCE` element of this *notify* message contains unique IDs of both, `ACTOR` and `ROOM`.

Operation `NotifyChangeOneWay` has the same parameters as `NotifyChange`, the only difference is that `NotifyChangeOneWay` does not have a return value. `NotifyChangeOneWay` can be used to send 'mission-uncritical' sensor data to the Context Server. This might apply to situations where data is frequently sent and the loss of one value in the meantime does not cause any undeterministic system behavior.

Following is an example for the `NotifyChange` parameter:

```
<?xml version="1.0" encoding="UTF-8"?>
<NOTIFY  CHANGETYPE="MODIFYDATA"
   CSTIME="2002-04-03T16:53:54.054"
   EVENTTIME="2002-04-03T16:53:53.001"
   INFORMATIONTYPE="RELATION" SENDERID="LocatedToRelation">
 <CONTEXT>
  <SUBCONTEXT TYPE="DYNAMIC">
   <CATEGORY TYPE="RelationName" VALUE="ACTOR in LOCATION"/>
   <CATEGORY TYPE="ContextInterpreter" VALUE="LocatedToRelation"/>
   <CATEGORY TYPE="BadgeID" VALUE="0e"/>
   <SUBCONTEXTELEMENT NAME="LOCATION">
    <SUBCONTEXTELEMENT NAME="ACTOR"/>
   </SUBCONTEXTELEMENT>
  </SUBCONTEXT>
  <SUBCONTEXT TYPE="DYNAMIC">
   <CATEGORY TYPE="RelationName" VALUE="ACTOR in LOCATION"/>
   <CATEGORY TYPE="ContextInterpreter" VALUE="LocatedToRelation"/>
   <CATEGORY TYPE="BadgeID" VALUE="0e"/>
   <SUBCONTEXTELEMENT NAME="ACTOR">
    <SUBCONTEXTELEMENT NAME="LOCATION"/>
   </SUBCONTEXTELEMENT>
  </SUBCONTEXT>
 </CONTEXT>
 <ONTOLOGY OID="82feb4d7-ec00-0000-0080-c6a1c64e5261"/>
 <XMLDATA>
  <![CDATA[
   <rdf:RDF>
    <rm:Relation name="ACTOR in LOCATION" transitiv="true">
     <rm:SDODataReference>
      <rm:SDODataContext>
       <CONTEXT>
        <SUBCONTEXT TYPE="STATIC">
         <CATEGORY TYPE="ActorID" VALUE="sta"/>
         <SUBCONTEXTELEMENT NAME="ACTOR">
          <SUBCONTEXTELEMENT NAME="GENERAL"/>
         </SUBCONTEXTELEMENT>
        </SUBCONTEXT>
       </CONTEXT>
      </rm:SDODataContext>
      <rm:SDODataUUID rdf:resource="D56FFAA8-6D13-11d4-B675-0010F3008056"/>
```

[27] This source info can be used to resolve what and why something has been performed by the system.

```
      </rm:SDODataReference>
      <rm:SDODataReference>
       <rm:SDODataContext>
        <CONTEXT>
         <SUBCONTEXT TYPE="STATIC">
          <CATEGORY TYPE="RoomID" VALUE="System"/>
          <SUBCONTEXTELEMENT NAME="LOCATION">
           <SUBCONTEXTELEMENT NAME="ROOM"/>
          </SUBCONTEXTELEMENT>
         </SUBCONTEXT>
        </CONTEXT>
       </rm:SDODataContext>
       <rm:SDODataUUID rdf:resource="6f2fcff6-ec00-0000-0080-926b024ab901"/>
      </rm:SDODataReference>
     </rm:Relation>
    </rdf:RDF>
   ]]>
  </XMLDATA>
  <SOURCE>
   <UUID>D56FFAA8-6D13-11d4-B675-0010F3008056</UUID>
   <UUID>6f2fcff6-ec00-0000-0080-926b024ab901</UUID>
  </SOURCE>
</NOTIFY>
```

4.2.1.3 SubscribeNotify Operation

The Context Server notifies all subscribed Context Interpreter and I-centric Services in case the subscribed context data has been changed. The XML parameter of the `subscribeNotify` operation is depicted in Figure 59.

The parameter for the `subscribeNotify` operation consists of the context categorization information (data to subscribe) and additional information. The context of the data subscribed is defined in the element `PARAMETER`.

The value of the parameter is the set of string values separated by '`/`'. The first value always represents the subcontext type (`static` or `dynamic` data). The following values define the subcontext element names. The parameter value `STATIC/Actor/General` subscribes `static` and `general` data of all actors (individual users).

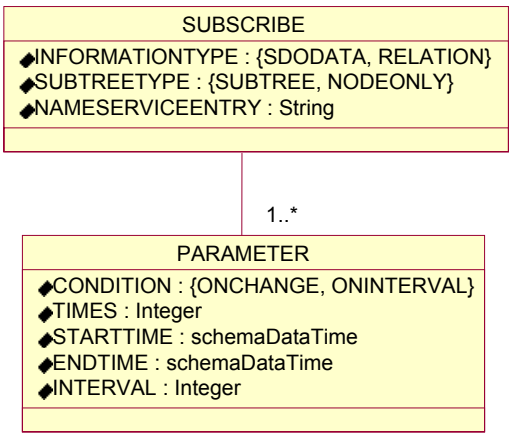


Figure 59: Parameter of the `subscribeNotify` operation

Attributes in the `PARAMETER` element defines a specific condition. Conditions for the subscription can be either '`ONCHANGE`' (send notification when the subscribed data has been changed), or '`ONINTERVAL`' (send notification in specified interval).

If the condition is not specified then by default the subscription is '`ONCHANGE`' basis. Further details of the subscription are specified with other attributes. All these attributes are optional except for the attribute '`INTERVAL`' that is accepted only for the condition '`ONINTERVAL`' for which it must be specified.

Name	Description	Value	Default
CONDITION	Condition for the subscription.	'ONCHANGE' or 'ONINTERVAL'	'ONCHANGE'
TIMES	Number of times the Context Server should send the notification when the subscribed data is changed.	Integer	Infinite
STARTTIME	When to start sending the notification when the subscribed data is changed.	String defining date & time in XML Schema dateTime format	When subscribed with Context Server
ENDTIME	End time for sending a notification when the subscribed data is changed.	String defining date & time in XML Schema dateTime formate	No end time
INTERVAL	Time interval in which subscribed data is expected if the condition is 'ONINTERVAL'.	Integer in seconds	No default value. Must be defined.

Table 3: Description of attributes defining conditions for the subscription

The attribute `INFORMATIONTYPE` defines to which profile the subscription is applied (Context Profile or Relation Profile). The `SUBTREETYPE` specifies, whether only changes in data of the specified context result in a notification or if all changes of this context and all sub nodes should be reported. For example, if a service subscribes for `static` data of the context `DEVICE` with subtreetype `nodeonly`, then changes in the context data categorization `DEVICE/ACTIVEDEVICE` will not be notified. However, if subscribed with subtreetype `SUBTREE` then it will be notified. At last, the NAMESERVICEENTRY holds the reference of the calling component, to which the Context Server will send any notification. The value of this attribute depends on the middleware technology used in a framework realization. The CallbackAddress identifies the calling component in a unique way that enables the Context Server to locate the referenced component. The current SDO implementation uses CORBA naming schemes[28].

For example:

```
SDOSystem.I-CentricServices.FacilityControl.Light
```

Below, a parameter is shown to subscribe all IR-Sensor data at the Context Server for a given interval. In the example, three types of context data is subscribed each with different conditions.

```
<?xml version="1.0" encoding="UTF-8"?>
<SUBSCRIBE INFORMATIONTYPE="SDODATA"
           NAMESERVICEENTRY=".ICCS.ContextInterpreter.condition"
           SUBTREETYPE="NODEONLY">
 <PARAMETER CONDITION="ONCHANGE"
            TIMES="5"
            STARTTIME="2002-03-25T15:00:00.000"
            ENDTIME="2002-03-25T14:00:01.000">
  DYNAMIC/Device/IRSensor
 </PARAMETER>
 <PARAMETER CONDITION="ONINTERVAL"
            STARTTIME="2002-03-27T15:00:00.000"
            ENDTIME="2003-03-27T17:00:00.000"
            INTERVAL="1">
  STATIC/ACTOR/GENERAL
 </PARAMETER>
 <PARAMETER>
  STATIC/LOCATION/ROOM
 </PARAMETER>
</SUBSCRIBE>
```

4.2.1.4 UnSubscribeAll Operation

UnSubscribeAll operation removes all beforehand subscribed parameters. The Context Server manages a list of which component has subscribed using which parameters.

[28] CORBA Naming Service [W-OMG-NS]

4.2.1.5 UnSubscribe Operation

UnSubscribe operation removes the specified subscription from the Context Server. The parameter of this operation is the same as of the subscribe operation. The Context Server returns an exception that the subscription could not be found when a component tries to unsubscribe a subscription that has not been subscribed before.

4.2.1.6 Ping Operation

Ping operation is used by SDOs or other components to verify whether the Context Server is still alive and can be accessed. Before initiating any communication with the Context Server other components should first call the method `ping()`. An exception while calling this operation indicates that the Context Server is not active. If no exception is raised the Context Server is running and can be accessed.

The following section introduces the *Resource Data Model*, which is applied to all information exchanged within the *Open Profiling Framework*. Especially, the Context Server deals with the management and storage of this information.

4.2.2 Resource Data Model

For the description of the information inside the *Open Profiling Framework*, the *Resource Data Model* is as important as the functional specification itself, since it shows how the data is represented and grouped, which is exchanged via the interfaces defined for all components of the framework. First, the profiles inside the Context Server, which store the information, are introduced. Secondly, the structure of the information is explained.

4.2.2.1 Context Profile, Relations Profile, and History Profile

The Context Profile stores all information about SDOs, individual preferences, and ambient information. It can be queried via Context Server' `getData` function.

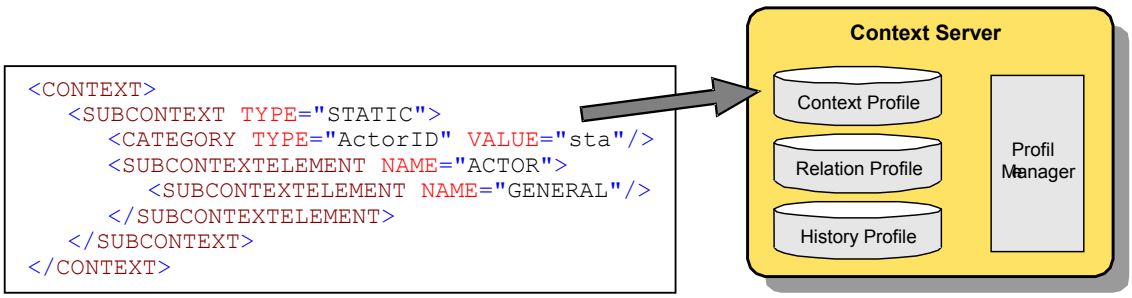


Figure 60: Example of a Context Profile Entry

The Relation Profile stores all information about relations between different entries in the Context database. E.g. when a user enters a room, it will be expressed with a relation between the user and the room.

For defining relations, the Resource Description Framework (RDF)[29] is used. RDF has been standardized by the W3C. It provides flexible mechanisms to define references between different resources. Therefore, RDF can be used to express the relationships between profile entries inside the Context Server.

[29] Resource Description Framework (RDF) [W-RDF]. For more information, see section 7.1.7.

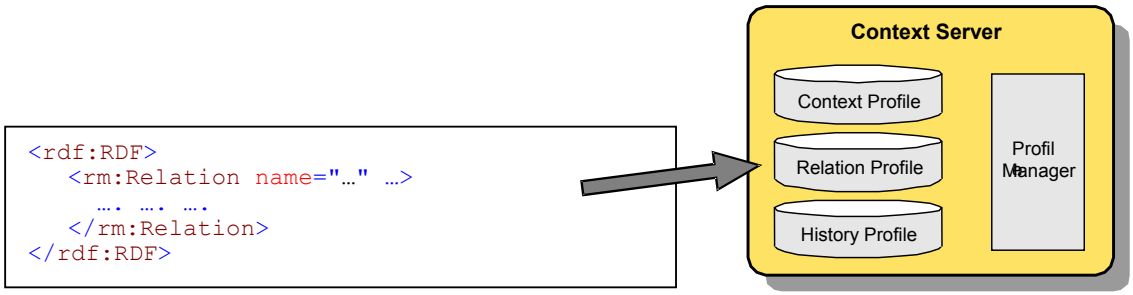


Figure 61: Example of a Relation Entry

The History database stores all changes of the Context Profile and Relation Profile. It can be used to restore any former state the Context Server had. For this reason, additional information about date/time is added to all changes regarding the Context Profile and Relation Profile. When context data is removed or modified in the *Context Profile* a copy is sent to the history profile. The history profile is mainly used for monitoring, maintenance, and automatic service creation as described in section 3.2.7.2.

4.2.2.2 Categorization of Data

The used data model for all information to be stored in *Contexts*, *Relations*, and *History*, orientates itself roughly on the Entity Relationship Model. Basic terms of this model are entities and relationships, as well as entity types and relationship types as their abstract forms. An entity represents an object of the real world, while the corresponding entity type forms the type of the object. Entities can have relations to each other, which are modeled through a relationship type between the corresponding entity types.

Applying these basic ideas of the Entity Relationship Model to the proposed framework, the following data model is suggested. All existing information inside the framework is divided into two types: **data** and **relations** between this data. The two information types for this separation are called *SDO data* and *Relation data*.

The **SDO data** type comprises information gathered or created by Super Distributed Objects. An abstract representation of SDO data that has been generated by a Context Interpreter belongs to this information type, too. SDOs represent the objects introduced in the vision for I-centric communications. For that reason, even individual users are modeled as SDO data. That enables to assign them to different individual communication spaces like all the other objects.

The **Relation data** type consists of information about relationships and constraints between certain SDO data type information. For instance, this information can describe the relation between an individual and a certain room, in which it is located, or the relation between an object and an individual user, which can represents ownership information for example.

Up to this point, the functional description of the framework has been completely independent from a specific standard or technology. When introducing the data model of the framework, this independency cannot be longer maintained. For the provision of an open and generic data model, certain features of the underlying technologies have to be represented in the model, too. Since the data model and the data representation technology are strongly bound together, a separation for reasons of a clear and divided specification would unnecessary restricts the capabilities of the framework. Therefore, all models introduced in the following are divided in two parts: an abstract information model and a technology driven resource data model.

The eXtensible Markup Language [XML][30] has been chosen as enabling technology for the framework, because it offers a standardized, open, and extensible data representation.

[30] For more details about XML, see section 7.1.6.

Both profiles, the Context Profile and Relation Profile, consist of XML typed data, which was sent by other components of the *Open Profiling Framework*. Every component in the open profiling framework that generates new information has to represent it in XML typed data structures before sending it to the Context Server. Therefore, each Profile can be seen as one XML file that holds the complete profile information for a certain moment in time.

The profiles inside the Context Server are accessed and manipulated by *Context Interpreters*, *Super Distributed Objects*, and *I-centric Services*. To provide a comfortable and efficient mechanism to access and manipulate the profile data, all profile entries have to be individually addressable and accessible.

The first step towards a useful structure of a profile is the consideration about the categorization of profile information. Since there are many different ways to identify a certain amount of data inside a profile, a method has to be found, which offers unequivocal addressing and searching of information inside profiles. The method introduced in the following combines a unique identification of data with a semantic categorization, which offers access to profiles based on both, identifiers and semantic descriptions.

Any data that is added to a profile at the Context Server gets an alphanumeric unique identifier assigned, which offers a direct access method. Additionally, categorization information is added to the data, which is provided partially by the sending component. This is necessary, because only the sending component, which is normally the creator of the data, has knowledge about the semantic of the data.

The sender ascribes a certain meaning to the symbols used in the categorization description. If another component wants to use this semantic description, it has to share the semantic of the terms the sender has initially used. In other words, these two components have to share a common ontology for the application domain of the respective profile information.

The categorization information is, like all information inside a profile, expressed in XML structures, too. Figure 62 shows the information model of categorization information.

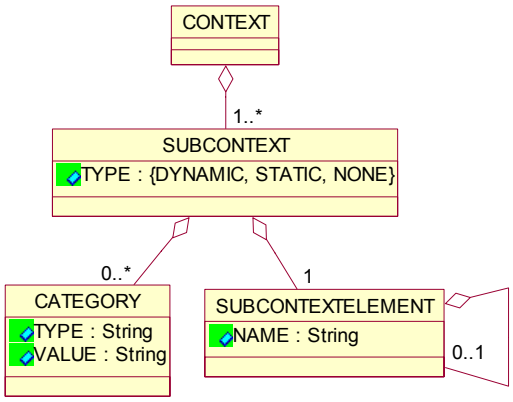


Figure 62: Categorization Model

All data provided by Super Distributed Objects or Context Interpreters belongs to a certain **Context**, to which the categorization refers. A Context can consist of one ore more **Subcontexts**, which represents the fact that a Super Distributed Object can be simultaneously part of several different communication spaces. For example, a keyboard can be part of a context that represents a computer system including monitor, mouse, and printer. On the other hand, that very keyboard can be part of a user's **active context** at the same time, if he is using this keyboard to write an email. Therefore, a Subcontext represents exactly one semantic description of the data. The attribute *Type* explains the kind of the data and can have two values: *static* or *dynamic*. Information belonging to a *static* Subcontext is more or less unlikely to change over time, while *dynamic* data is only valid for a short period and will change as soon as new data arrives at the sensing component. The Context Server will automatically delete all *dynamic* pro-

file information that exceeds a certain, configurable lifetime. A sensor can for instance provide data of the type *static* to the Context Server, which holds information about the position of the sensor, the manufacture, or the like. Actual sensory information about the sensor's environment is sent to the Context Server with the type *dynamic*.

Since the Subcontext has the purpose to describe the data itself, a model for this description is applied, that meets the requirement of identification. The used model consists of an ordered list of **Subcontext Elements** that have a *Name* and additional **Categories**. The list shall be used to describe the data increasingly detailed, starting with a general term, and ending with a detailed and explicit one. For example, a telephone can have the following list of Subcontext Elements: *Device → Telecommunication Device → Telephone → Mobile.*

Categories are used to specify further a certain Subcontext. They have two strings as parameter: a *Type* and a *Value*. A Category for the above-mentioned example can hold the information, that it is an ISDN telephone (*Type* = 'telephone type' and *Value* = 'ISDN'). An object can have several categories because it can pertain to different contexts.

Since profile information is expressed as XML typed data, the categorization of profile information can be given as XML data, too. Therefore, the just introduced information model for profile categorization is mapped to a XML *Schema*, which acts as a technology-driven data model specification. More information about XML Schema is given in section 7.1.10.

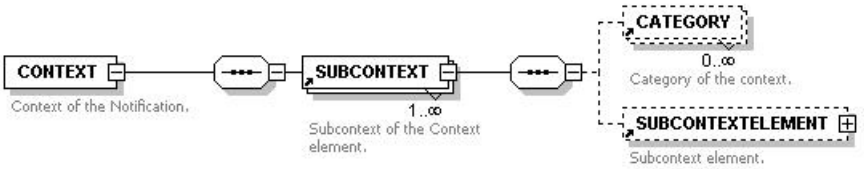


Figure 63: Context Element XML Schema Specification

Figure 63 shows the context schema as a diagram. It defines the structure of XML typed data that represents a profile information categorization. This schema is an XML representation of the information model shown in Figure 62. Objects and attributes in the information model have been mapped to XML elements with attributes. The aggregation relationships between the objects have been expressed as a containment relation in the XML Schema with the corresponding element-number restrictions.

The example below shows categorization information that represents a Context with only one static Subcontext. The Subcontext contains an ordered list of two Subcontext Elements: 'ACTOR' and 'GENERAL'. The SUBCONTEXT element has a *category* defining it of *type* ActorID with the value sta. The ordered list is represented as containment relation in XML: The head element contains its successor element.

```
<CONTEXT>
  <SUBCONTEXT TYPE="STATIC">
          <CATEGORY TYPE="ActorID" VALUE="sta"/>
          <SUBCONTEXTELEMENT NAME="ACTOR">
                  <SUBCONTEXTELEMENT NAME="GENERAL"/>
          </SUBCONTEXTELEMENT>
  </SUBCONTEXT>
</CONTEXT>
```

Section 7.2.3.1 shows the complete XML Schema specification of the context categorization. Any information that is to be inserted to any profile, managed by the Context Server, is evaluated against this specification. Information that is not compliant to the XML Schema is discarded by the Context Server.

The categorization for a given profile information shown in this section is used as a semantic description, which offers framework components to locate and to identify relevant data. The next section describes how this categorization can be additionally used for structuring a complete profile like the Context Profile.

4.2.2.3 Context Tree

Every XML typed data can be seen as a tree with XML tags as nodes and leafs. Therefore, the example for categorization information in the previous section can be seen as a tree, too. Figure 64 shows such an XML tree, where the tags (*Context, Subcontext, Subcontext Element,* and *Category*) are nodes and the attributes (*Name, Type,* and *Value*) are attached to them. The tree diagram is just a different visual representation of XML data. Both show the same information.

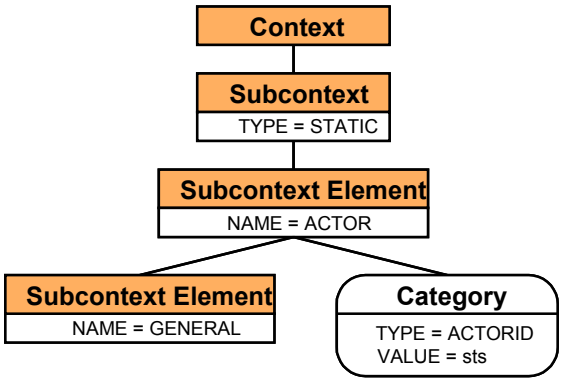


Figure 64: Categorization Example as XML Tree

Since the structure of all profiles conforms to the XML Schema defined, the categorization of profile information is used to retrieve the data when required. To store the profile information, the Context server encapsulates the data with its categorization information and adds it to the corresponding profile. The following example illustrates this. The example shows how the profile information of an *Actor* (individual user) described above is stored in the *context profile.*

```
<DATA CHANGETYPE="INSERTDATA" CSTIME="2002-04-11T10:56:22.022"
      dataUUID="D56FFAA8-6D13-11d4-B675-0010F3008057" SOURCE="initial_data">
  <SUBCONTEXT TYPE="STATIC">
   <CATEGORY TYPE="ActorID" VALUE="sta"/>
   <SUBCONTEXTELEMENT NAME="ACTOR">
    <SUBCONTEXTELEMENT NAME="GENERAL">
     <DATAELEMENT CSTIME="2002-04-11T10:56:22.022"
                  EVENTTIME="2002-04-11T10:56:20.012"
                  SOURCE="initial data"
                  rdf:about="D56FFAA8-6D13-11d4-B675-0010F3008057">
      <CONTEXT>
       <SUBCONTEXT TYPE="STATIC">
        <CATEGORY TYPE="ActorID" VALUE="sta"/>
        <SUBCONTEXTELEMENT NAME="ACTOR">
         <SUBCONTEXTELEMENT NAME="GENERAL"/>
        </SUBCONTEXTELEMENT>
       </SUBCONTEXT>
      </CONTEXT>
      <ONTOLOGY OID="H56FFAA8-6D13-11d4-B635-0010F3048053"/>
      <XMLDATA>
       <Actor ID="sta">
        <SurName value="Arbanowski"/>
        <FirstName value="Stefan"/>
        <Description value="Stefan Arbanowski"/>
       </Actor>
      </XMLDATA>
     </DATAELEMENT>
    </SUBCONTEXTELEMENT>
   </SUBCONTEXTELEMENT>
  </SUBCONTEXT>
 </DATA>
```

It can be seen in the example above that the data starts with the `DATA` element. The profile information contains additional information. The Context Server stores all profile information with a unique ID (attribute `dataUUID`). This unique identifier is also used to retrieve the data directly without going through all the categorization information.

The time when the Context Server received the profile information is stored in the attribute `CSTIME`. This time conforms to the `dateTime` format specified by the XML Schema specification. The source information in the attribute `SOURCE` stores the identification of the sending component.

Proper XML XPATH statement created using this categorization information is used to query specific data. The use of XML XPATH statement makes it easier to search through all data in the profile.

A relation connects two or more profile information elements with each other. A relation has a specific name (attribute `Name`), it can be transitive or not (attribute `Transitive`), and it has additional characteristics, which are stored in an optional `XMLDATA` element.

The connection of profile elements is realized with `REFERENCE` elements, whereas each of these element points to a profile element of the intended relationship. A reference holds the identifier and the categorization of a profile element.

The transitivity attribute is important for automatic reasoning processes. The term transitive belongs to the area of formal logic where it specifies the following reasoning process: If the implications ($A \Rightarrow B$) and ($B \Rightarrow C$) are true, then the transitive reasoning results it the fact that A also implies C ($A \Rightarrow C$).

Implication 1: $(A \Rightarrow B)$

Implication 2: $(B \Rightarrow C)$

Transitive Reasoning: $(A \Rightarrow B) \wedge (B \Rightarrow C) \Rightarrow (A \Rightarrow C)$

This formal transitive reasoning can be applied to Profile Evaluation processes. If a relationship can be processed with transitive reasoning, then it has the transitive attribute value *true*. For example, there exist a relationship between an actor (A) and a certain room (B) as well as a relationship that connects this room (B) with a certain building (C).

If both relations are transitive, it can be derived that there is also a relationship between the actor (A) and the building (C). Relationships that cannot be processed in this way have the value *false* in their transitive attribute.

The following example illustrates the introduced relationship information model. The example shows the content of the `XMLDATA` element of profile information that is stored in the Relation Profile.

```
<XMLDATA>
 <rdf:RDF>
  <rm:Relation Transitiv="true" name="ACTOR in LOCATION">
   <rm:SDODataReference>
    <rm:SDODataContext>
     <CONTEXT>
      <SUBCONTEXT TYPE="STATIC">
       <CATEGORY TYPE="ActorID" VALUE="sta"/>
       <SUBCONTEXTELEMENT NAME="ACTOR">
        <SUBCONTEXTELEMENT NAME="GENERAL"/>
       </SUBCONTEXTELEMENT>
      </SUBCONTEXT>
     </CONTEXT>
    </rm:SDODataContext>
    <rm:SDODataUUID rdf:resource="719C511C-4C40-48fe-9ACF-1224B1DF0C0E"/>
   </rm:SDODataReference>
   <rm:SDODataReference>
    <rm:SDODataContext>
     <CONTEXT>
      <SUBCONTEXT TYPE="STATIC">
       <CATEGORY TYPE="RoomID" VALUE="5030"/>
       <SUBCONTEXTELEMENT NAME="LOCATION">
        <SUBCONTEXTELEMENT NAME="ROOM"/>
       </SUBCONTEXTELEMENT>
      </SUBCONTEXT>
     </CONTEXT>
    </rm:SDODataContext>
    <rm:SDODataUUID rdf:resource="2C933718-CF6A-4f70-AB37-9F780B427709"/>
   </rm:SDODataReference>
  </rm:Relation>
 </rdf:RDF>
</XMLDATA>
```

The relationship describes that an actor is located in a certain location. It connects profile information of the categorization `/ACTOR/GENERAL/` with profile information of the categoriza-

tion /LOCATION/ROOM/. The RELATION element contains two attributes and an additional XMLDATA element for further attributes, descriptions, or the like. Two REFERENCE elements contain the categorization and the unique identifier of the profile entities they refer to.

4.2.2.4 Context Tree Setup

Considerations about a suitable profile structure should follow a top-down approach: The development of a general layout of the expected profile information should be done first before going into a detailed specification later. To keep the openness of the framework, all advices given in this section have a general character. Detailed recommendations would unnecessary fix the structures inside the profiles.

The following four questions should be contemplated while developing a structure for the profile information.

- which profile information should be represented as SDO-data and which as a relationship between SDO-data elements? (General design criteria)
- how should the profile information be structured inside the profiles? (General profile structure)
- how should information be represented that belongs together? (Decision of context aggregation)
- which information should be added to the categorization? (Decision of necessity)

Each of these topics will be discussed in an own section in the following.

Super Distributed Objects and Relationships

The first decision should cover the issue, which profile information should be represented as SDO-data entity and which as a relationship between these entities. This topic belongs to the group of general modeling questions that deeply influence the complete profiling framework. Since there is a large amount of possible relationships for each object of the real world, a decision has to be made, to which extend they should be represented in profile-relations. It is a question of feasibility and necessity. In most cases, it is sufficient to store profile information of a room for instance, only with relationships to the building it is located in as well as relationships to devices and humans that are located in it. The fact that a certain architect has planned this special room or that the worker ‘xyz’ has built the walls, is irrelevant for most application domains.

It has to be kept in mind that relationships will be used for reasoning. For example, if an individual is located in a certain room, there will be a relation between the individual and the room. An automatic evaluation process now can reason from the relationship between the room and the actor and the relationship between the room and the building, that the actor also has a relation to that building. Obviously, such automatic reasoning processes are only possible, if the chosen information model provides all relevant relationships. It is therefore important to find a good compromise for this design decision that keeps all necessary relationships between profiles disregarding unnecessary ones.

General Profile Structure

The XML-tree structure of the Context Profile and the Relation Profile cannot be directly prescribed, since it is build up during the runtime of the framework. It evolves as a sum of all categorizations from inserted profile information and is therefore not predictable. Nevertheless, it can be influenced through general recommendations for the creation of categorization, which profile structure will exist in the end.

In general, all categorization for profile information should use meaningful and unique terms for the description of Subcontext-Elements and Categories. In any case, the semantic meaning of the respective name should be specified in an appropriate ontology, but meaningful terms avoid

misunderstandings from the start and help developers to identify the appropriate profile information for their components.

The ordered list of Subcontext-Elements should form a hierarchical description of the corresponding profile information. It should start with general and abstract terms that define the topmost class of categorization and assign the data to certain groups of information. These topmost Subcontext-Elements building the main branches of the profile-tree and should be selected in respect to the application domain. If the application domain comprises the telecommunication environment of a company building for instance, elements like *Individual*, *Device*, and *Location* can be feasible.

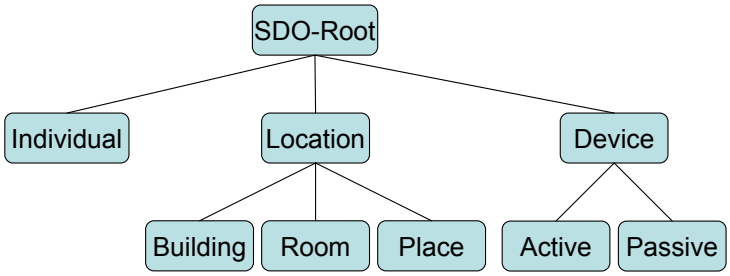


Figure 65: Tree Structure Recommendation Example

After the first Subcontext-Element in a categorization, the used terms should become more specific and end up in a very specialized term, which identifies the precise type of the profile information. The *Location* element for example, can be followed from elements like *Building*, *Room*, or *Place*, while the *Device* element is followed by an *Active* and a *Passive* element. The resulting first branches of a context tree regarding the building example are shown in Figure 65.

Context Aggregation

The third point of the list is concerned with context aggregation. Some data inside the profile belongs together, i.e. information gathered about a specific individual. Such an aggregation can be modeled using relationships between the entities that describe their unity. However, if a component wants to retrieve some information about an individual, it has to process all these relationships to get the requested data. This approach can be used for aggregation, but it might cause complex evaluation mechanisms, because a potential big number of relations have to be resolved and evaluated.

A different and faster method is to use categories for context aggregation additionally to relationships. Every individual is assigned to a unique identifier like a username or ID. Now, profile information belonging to a specific actor will be added a category element that has the type *ActorID* for instance and holds the corresponding actor identifier. If a component wants to get all profile information about an individual, it just has to request all data in the actor-subtree that has a category with the appropriate *ActorID* value. Since this method can be used parallel to the relationship-approach, it is possible to use both methods in parallel to leave the decision, which one to use, to the requesting component.

Categorization

The categorization of a given profile information consists of two major parts: the list of Subcontext-Elements and the Categories attached to these elements. The Subcontext-Elements offer a general semantic description of the profile information that identifies its position inside the context tree. The Categories are describing these elements more detailed by assigning certain type-value pairs to them. Which information should be added with Categories to the categorization is a matter of choice, a matter of feasibility, and a matter of the intended application domain.

A compromise has to be found, that lies between the two extremes of using no Categories at all and of inserting unnecessary information. When using no Categories at all, the categorization is not exact and too vague. On the other hand, if nearly all information from the actual profile is

additionally added to the Categories, it will be difficult to find relevant information and the profile will be highly redundant. A good rule of thumb is, to add only metadata about the profile information to the categorization. The class of a telephone for example is important for the categorization (ISDN or analog), while its color and production-date is not. The actual profile information has to be stored in the XML-data section of the profile.

Static versus Dynamic Data

Different kinds of information for a given Super Distributed Object component will be stored inside the Context Profile. There will be information about the current state of the SDO or actor in the real world as well as metadata that describe the general info about an SDO. These two types of data have to be separated to make them individually accessible. The question is how these two types should be marked using categorization.

The categorization-model provides a method for labeling profile information as *static* and *dynamic*. These two types, which are stored in the Sub-context of the categorization, can be used to provide the separate identification of the profile information and metadata about the Super Distributed Object.

The terms *static* and *dynamic* have been chosen, because the meaning of metadata is too vague and subjective to be used for the separation. There can be metadata about the information the Super Distributed Object has gathered as well as metadata about the SDO itself. Information gathered by an SDO can be processed to metadata by a Context Interpreter and stored again. Since the Context Server is able to store nearly every kind of information provided by Super Distributed Objects or Interpreting Components, the differentiation with use of the term metadata would be inapplicable.

The Context Server itself does not concern about the abstraction level of the stored profile information. It is more relevant for profile storage, if the data will be changed very often. The separation-decision is therefore made with respect to the proposed lifetime of the profile information: If it is dynamical information about the current state of the Super Distributed Object that is likely to change during the lifetime of the SDO, then it is labeled as *dynamic*.

This can be information from a sensor or microphone for example, which is changing very often. Is the profile information just stored once and if it is very unlikely that it will change during the lifetime of the SDO, then it is labeled as *static*. For instance, the personal information about an individual, which comprises his name, his gender, and his address, will be stored as *static*.

The information that the profile contains metadata has to be stored as additional information inside the actual XML data or in the Categories of the categorization.

4.2.2.5 Guidelines for Structuring Information

The Open Profiling Framework offers a profiling mechanism that can cope with any kind of profile structure. On the one hand, this is certainly an advantage, since changed profile structures and new information or devices can be introduced without hesitation. The generic approach of the framework has been developed with the clear intention that an I-centric communications system is always subject to minor and mayor changes. On the other hand, the lack of fixed and predefined structures is a disadvantage, too.

A clear and lucid structure is essential when dealing with such large amounts of heterogeneous and variable data that arises in an I-centric communications system. The power of such a system results from the fact, that different components have access to gathered profile information *without* the prior knowledge of their existence. Therefore, a wrongly chosen or evolved profile structure that hides information because of inappropriate categorizations leads to an ineffective system.

To unleash the potential power of open profiling, the uncertainty about structures has to be slightly restricted. The key point of open profiling in the proposed framework can be seen in the

semantic categorization of profile information. The categorization influences not only the access to profiles. Since the profile structure is build with use of the categorization, it additionally forms these evolving structures. It is therefore important to develop guidelines for an intelligent categorization that leads to clear profile structures.

Usage of XML

The first consideration about structures is concerned with the usage of XML. Since nearly everything in this framework is expressed in XML, it seems appropriate to offer a few general recommendations. Because of the flexibility of XML, data structures can be stored in many different ways, always representing the same information. The following example shows three different representations of the same piece of data, a date.

Structure 1	Structure 2	Structure 3

`<date value = "12/11/00"/>`	`<date>12/11/00</date>`	`<date>`
		`  <day>12</day>`
		`  <month>11</month>`
		`  <year>00</year>`
		`</date>`

Table 4: Data Representation in XML

The first variant stores the date as a complete text string in an XML-attribute. The second variant also stores the date as a complete text string, but in an XML-element as character-data. The last variant stores the parts of the data in an element structure with character-data, too. An important decision about data representation to chose between data storage in attributes and data storage in child elements. Attributes are easy and fast to access, since they are directly attached with a name to an element. Normally, attributes are also smaller in character-usage as child elements, as the example shows: Structure 3 needs about three times more character space than Structure 2. However, there are also several problems, which occur when using attributes:

- Attributes cannot contain multiple values. (Child elements can)
- Attributes are not easily expandable. (For future changes)
- Attributes cannot describe structures (Child elements can)
- Attributes values are not easy to validate against a XML Schema.

There are no hard rules about when to use child elements and when to use attributes. It is a matter of personal choice and of the intended application domain. A good guideline is that the data itself has to be stored in elements and information about the data (meta-data) has to be stored in attributes. Additionally, attributes should only be used for atomic values like number values or short text strings, since structures can only be expressed very restricted inside them.

If attributes are used as containers for the actual data, the resulting XML documents will be very difficult to read and (that is more important) also very difficult to maintain and change. On the other side, attributes can be retrieved faster than child elements when using an XSL Transformation or an XML-parsing application. When processing large amounts of XML-data, it can be therefore usable to store the character data partially in attributes to increase processing speed.

4.2.3 Context Server Maintenance Tools

As the Context Server manages all preferences, ambient information, and SDO states, an increasing amount of data is the result after running an I-centric system for a while. During the implementation phase a lot of effort has been spend to reduce the data that has to b stored persistently:

- dynamic data is moved to the history profile after exceeding its valid period
- removing SDO data from the Context Server removes all relations assigned to that data
- temporary system data used for I-centric Services is removed after service execution
- a consistency mechanism checks all touched data and removes it when inconsistent

Nevertheless, some times it is necessary to maintain the data stored in the Context Server by hand (e.g. to remove SDO data of SDOs which did not deregister correctly). For that reason, two maintenance tools have been implemented to manipulate Context Server data directly.

Figure 66 shows the **Relation Creator** and the **Relation Manager**. The Relation Creator allows creating relations between different SDO data entries. The relations, which can be assigned to SDO data, have to be specified and registered with the Ontology Server beforehand.

The Relation Creator does a lookup on the Ontology Server and provides the list of relations available for creation (see circle in Figure 66). Furthermore, the Relation Creator allows creating new relations to be registered with the Ontology Server.

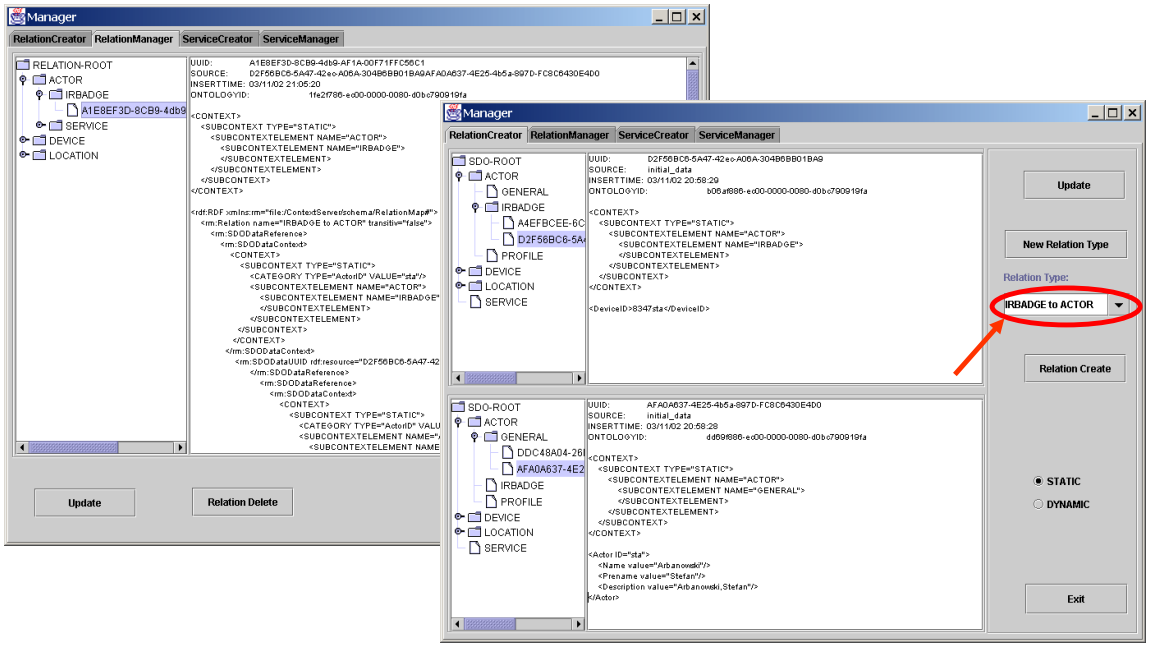


Figure 66: Context Server Maintenance - Relation Creator and Relation Manager

The Relation Manager is mainly used to check or delete existing relationships. They can be viewed ordered by several criteria (e.g. originating data, target data, time, and context).

Both, the Relation Creator and the Relation Manager are only designed for system administrators. A user of an I-centric communications system does not have to use or understand these tools. Moreover, the I-centric runtime-system does not require for such maintenance activities. Creating relationships is normally the task of Context Interpreters.

4.3 Ontology Server

In order to enable the integration of large-scale distributed objects and to allow services to handle them flexible an Ontology Server stores the syntax and semantics of information that is generated by these diverse objects.

An Ontology Server stores the interface definition and description of the semantic of controllable SDOs and provides validation and mapping of different ontologies. An SDO has to register its ontology with the Ontology Server. The Ontology Profile managed by the Ontology Server can be accessed directly, i.e. other components can add, modify, and delete ontologies.

4.3.1 Ontology Server Interface

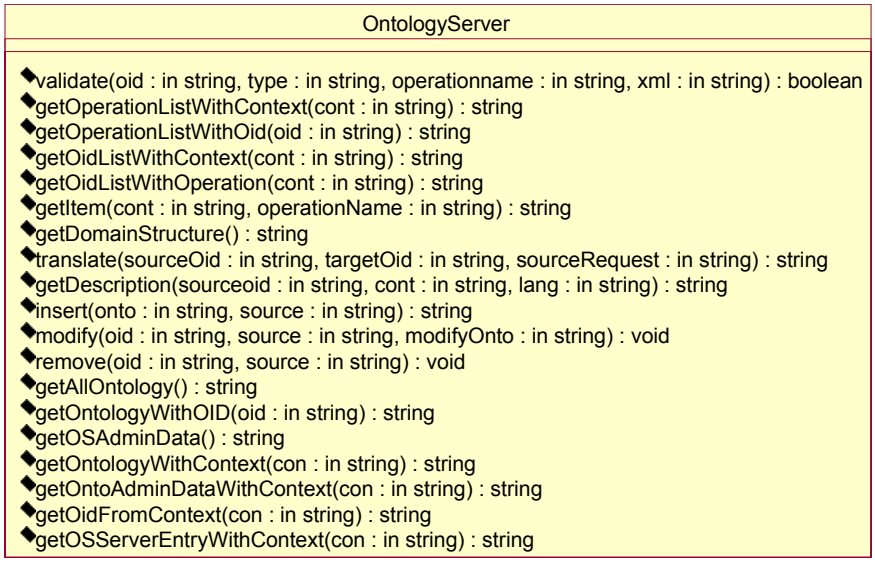


Figure 67: Interface of the Ontology Server

The Ontology Server interface offers operations that give access to the Ontology Server's functionality, which follows the ontology concept of the FIPA organization (see Figure 67).

Ontology Server interface	
Operation	Description
`validate`	checks whether the syntax of data is correct. Data can be an SDO request / notify, an I-Centric Service request / notify or a context relation.
`getOperationListWithContext`	returns a list of all operations, which is found for given *context*
`getOperationListWithOid`	returns a list of all operations, which is found for given `OID`
`getOidListWithContext`	returns a list of all `OIDs`, which are found with the given context
`getOidListWithOperation`	returns a list of all `OIDs`, which support the requested operation.
`getItem`	gets a certain entry of the ontology information
`getOperation`	gets the ontology of a certain operation
`getDomainStructure`	gets the structure of the ontology domain
`translate`	translates the *sourceRequest* to the *targetRequest*
`getDescription`	gets the description of the given context from ontology with defined oid.
`insert`	inserts the given ontology in the Ontology Server
`modify`	exchanges the specified ontology with a new one, if the new one is valid.
`remove`	removes the defined ontology. If a link to this ontology exists from another ontology, this ontology will not be removed
`getAllOntology`	gets all Ontology Server entries (*domain data*, *ontology entries*, *administration data*)
`getOntologyWithOID`	gets the ontology of given `OID`
`getOSAdminData`	gets the administration data of Ontology Server, like up-time, number of registered ontologies, or number of queries since started
`getOntologyWithContext`	gets the ontology of the given context. Context means the unique key, which it consists of `CONTEXT+ONTOLOGYNAME+VERSIONNUMBER`
`getOntoAdminDataWithContext`	gets the administration data of an ontology of the given context. Context means the unique key (`CONTEXT+ONTOLOGYNAME+VERSIONNUMBER`)
`getOidFromContext`	gets the `oid` to the given context. Context means the unique key, which it consists of `CONTEXT+ONTOLOGYNAME+VERSIONNUMBER`
`getOSServerEntryWithContext`	gets the Ontology Server entry with given context. Context means the unique key (`CONTEXT+ONTOLOGYNAME+VERSIONNUMBER`).

Table 5: Operations of the Ontology Server interface

4.3.2 Ontology Definition

The behavior and description of the components are defined by their ontology. The ontology description of a component is an XML document conforming to the ontology XML Schema. Figure 68 describes the structure of the ontology inside the *Open Profiling Framework*.

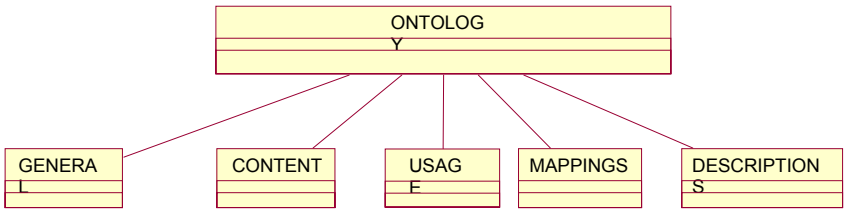


Figure 68: Ontology structure

As seen in Figure 68, ontology contains five different types of information: `GENERAL`, `CONTENT`, `USAGE`, `MAPPINGS`, and `DESCRIPTIONS`. The first four are the main parts. The `DESCRIPTIONS` part is an annotation to the ontology. `DESCRIPTIONS` help to explain functionality of operations or semantic of the ontology data.

4.3.2.1 General

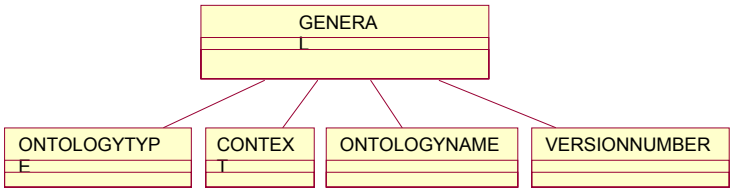


Figure 69: Ontology Structure: GENERAL

The `GENERAL` data (see Figure 69) contains the information to classify and to group ontologies. The `ONTOLOGYTYPE` contains the type of an ontology, for instance: [SDO | context relation | I-Centric Service]. The field `CONTEXT` is used to define the domain of the ontology. The syntax of domain description is similar the syntax of description for a directory, for instance 'home_control/device/light' or 'home_control/device/music'.

The `ONTOLOGYNAME` is a unique key to name an ontology. In addition, the `VERSIONNUMBER` denotes the version of an ontology. The fields *context*, *ontology name* and version number build a key (`CONTEXT+ONTOLOGYNAME+VERSIONNUMBER`), which is unique for every ontology.

```
<GENERAL>
  <ONTOLOGYTYPE>SDO</ONTOLOGYTYPE>
  <CONTEXT>/Home/SDO/Conditions/Light</CONTEXT>
  <ONTOLOGYNAME>DIMMER</ONTOLOGYNAME>
  <VERSIONNUMBER>1.0</VERSIONNUMBER>
</GENERAL>
```

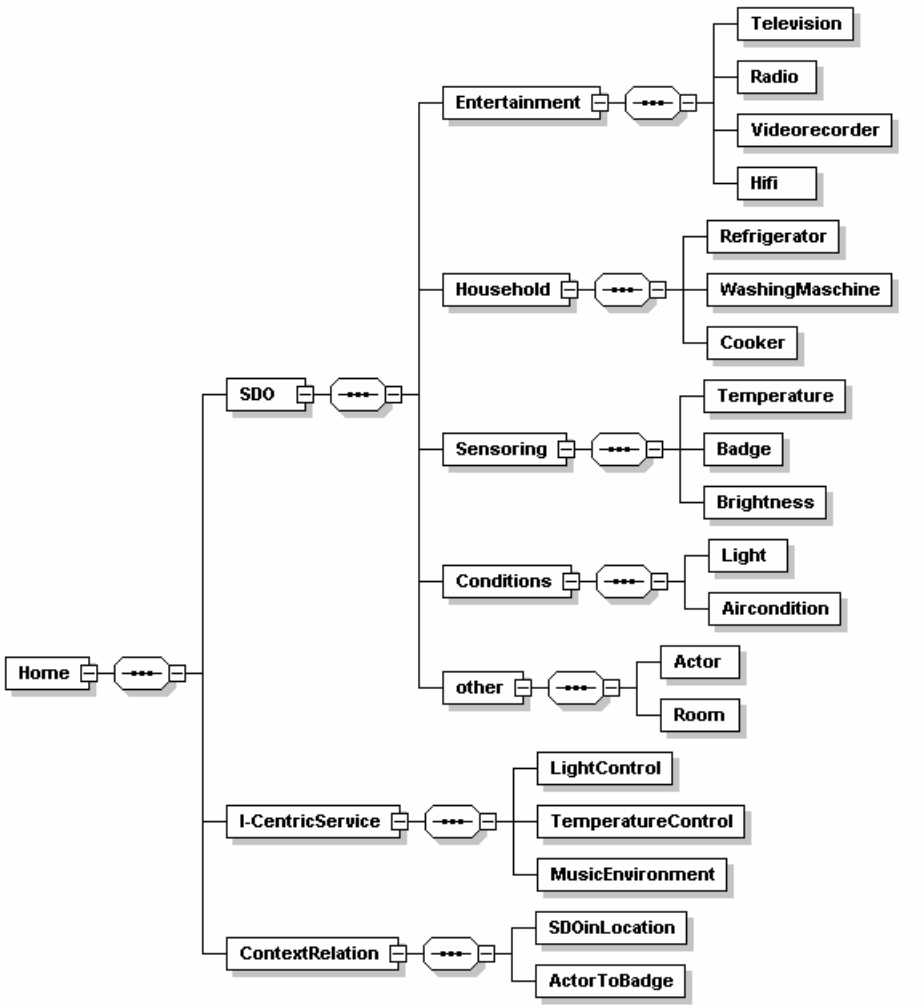


Figure 70: Ontology Structure Example: Domain Context

The context can be viewed as a tree. As an example, Figure 70 represents a home control domain. Devices of a home can assign to these categories respectively *contexts*, for instance the television from living room to `'Home/Entertainment/Television'`. These contexts make it easier for a user of the interactive service creation to find the required functionality, to be embedded in an I-Centric service.

4.3.2.2 Content

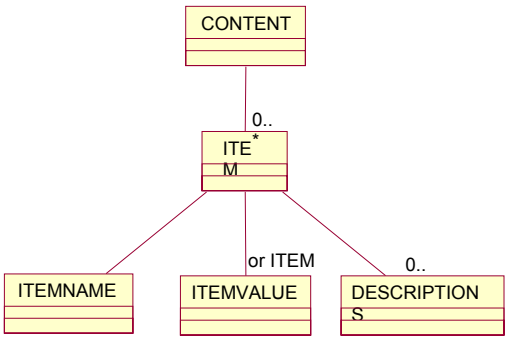


Figure 71: Ontology Structure: CONTENT

The content data (Figure 71) contains information about the structure of context relations, state of an SDO or state of an *I-centric Service*. This data structure is constructed recursively, to build flat or complex structured data. The field `ITEM` consists of an `ITEMNAME` field, an

ITEMVALUE or an included ITEM structure and a DESCRIPTION structure, which is optional. DESCRIPTION explains the ITEM structure. The field ITEMNAME names the item structure. The ITEMVALUE describes the item values, which are allowed to assign to this item.

```
<CONTENT>
  <ITEM>
   <ITEMNAME>UUID</ITEMNAME>
   <ITEMVALUE>
    <xs:simpleType>
     <xs:restriction base="xs:string"/>
    </xs:simpleType>
   </ITEMVALUE>
   <DESCRIPTIONS>
    <DESCRIPTION>
     <LANGUAGE>EN</LANGUAGE>
     <LONGDESCRIPTION>
      This filed contains a unique key for this SDO
     </LONGDESCRIPTION>
     <SHORTDESCRIPTION>UUID</SHORTDESCRIPTION>
     <PICTURE/>
    </DESCRIPTION>
   </DESCRIPTIONS>
  </ITEM>
 </CONTENT>
```

The possible values are defined through types in the specification language XML Schema, because it can define types exactly and in different manner. Besides, many tools and APIs exist to validate and evaluate documents conform to this language.

4.3.2.3 Usage

The USAGE data contains information about the usage of *SDO*s and *I-Centric Services*. There are two types of data, OPERATION and OPERATIONLINK. The usage data can contain several instances of these types. The operation type describes an operation and the operation link is only a link to another operation. An operation link means that an SDO, which have a link to an operation of another SDO operation in their ontology, support the same operation like the other SDO.

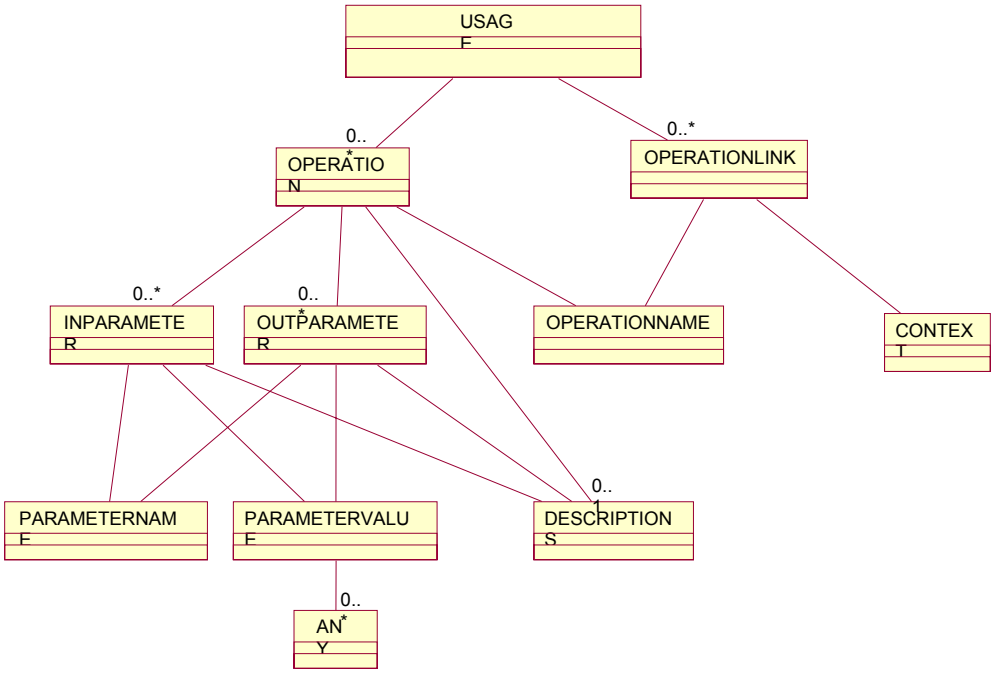


Figure 72: Ontology Structure: USAGE

An operation link consists of a *context* field and a field, which contains the name of the supported operation. The *context* here means a unique key to the source ontology. It is built from the general data fields CONTEXT, ONTOLOGYNAME, and VERSIONNUMBER in the following manner CONTEXT+"/"+ONTOLOGYNAME+"/v" + VERSIONNUMBER, for instance "Home/Entertainment/Television/Hitachi/v1.0".
An OPERATION structure is more complex than an OPERATIONLINK. It consists of an OPERATIONNAME field, INPARAMETER structures, OUTPARAMETER structures and an optional DESCRIPTIONS structure. The OPERATIONNAME is an identifier for the OPERATION structure. The INPARAMETER and OUTPARAMETER match the operation parameter and return values similar in programming languages. Both are from type PARAMETER that means they have the same structure. The PARAMETER type consists of a PARAMETERNAME field and PARAMETERVALUE structure, which contains XML Schema statements to define the value. The DESCRIPTIONS structure is optional. It can give explanation about the specific IN-/OUTPARAMETER.
In the following, an USAGE part is shown. It illustrates a 'Dimmer' that provides two operations (Dimm and ChangeState). Dimm is specified in the usage part directly, whereas ChangeState is specified by referring to the ontology of another component (Light).

```
<USAGE>
  <OPERATION>
   <OPERATIONNAME>Dimm</OPERATIONNAME>
   <INPARAMETER>
    <PARAMETERNAME>DimmValue</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:int">
       <xs:minInclusive value="0"/>
       <xs:maxInclusive value="100"/>
      </xs:restriction>
     </xs:simpleType>
    </PARAMETERVALUE>
    <DESCRIPTIONS>
     <DESCRIPTION>
      <LANGUAGE>EN</LANGUAGE>
      <LANGDESCRIPTION>
       The Dimm Value gives the desired brigthness of the
       dimmer (0 - light off 0% ... 100 - light on 100%)
      </LANGDESCRIPTION>
      <SHORTDESCRIPTION>DimmValue</SHORTDESCRIPTION>
      <PICTURE/>
     </DESCRIPTION>
    </DESCRIPTIONS>
   </INPARAMETER>
   <OUTPARAMETER>
    <PARAMETERNAME>Dimmed</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:int">
       <xs:minInclusive value="0"/>
       <xs:maxInclusive value="100"/>
      </xs:restriction>
     </xs:simpleType>
    </PARAMETERVALUE>
    <DESCRIPTIONS>
     <DESCRIPTION>
      <LANGUAGE>EN</LANGUAGE>
      <LANGDESCRIPTION>
       The Dimmed Value is the current brigthness of the
       dimmer (0 - light off 0% ... 100 - light on 100%)
       </LANGDESCRIPTION>
      <SHORTDESCRIPTION>DimmedValue</SHORTDESCRIPTION>
      <PICTURE/>
     </DESCRIPTION>
    </DESCRIPTIONS>
   </OUTPARAMETER>
   <DESCRIPTIONS>
    <DESCRIPTION>
     <LANGUAGE>EN</LANGUAGE>
     <LANGDESCRIPTION>
      Operation Dimm set the dimmer of given value
     </LANGDESCRIPTION>
     <SHORTDESCRIPTION>Operation Dimm</SHORTDESCRIPTION>
     <PICTURE/>
    </DESCRIPTION>
   </DESCRIPTIONS>
```

```
  </OPERATION>
  <OPERATIONLINK>
        <CONTEXT>/Home/Conditions/Light/LIGHT/v1.0</CONTEXT>
   <OPERATIONNAME>ChangeState</OPERATIONNAME>
  </OPERATIONLINK>
</USAGE>
```

4.3.2.4 Mappings

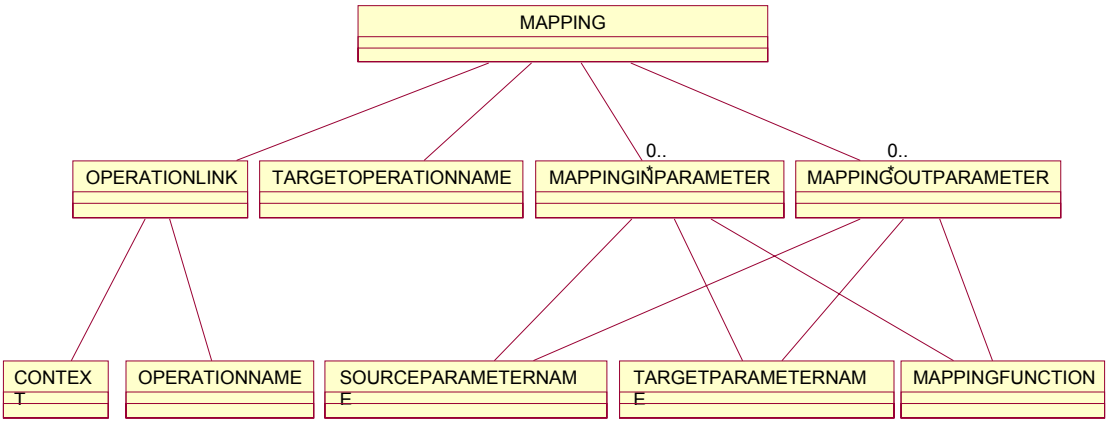


Figure 73: Ontology Structure: MAPPINGS

MAPPING provides the information to map between different ontologies. The `MAPPINGS` can contain several `MAPPING` structures, which consists of an `OPERATIONLINK` structure, `TARGETOPERATIONNAME` field, `MAPPINGINPARAMETER` structure, and `MAPPINGOUTPARAMETER` structure.
The `OPERATIONLINK` refers to the operation in another ontology if an operation in the current ontology offers the similar functionality. The structures `MAPPINGINPARAMETER` and `MAPPINGOUTPARAMETER` contain the `SOURCEPARAMETERNAME` and the `TARGETPARAMETERNAME`, which define the corresponding parameters of the operations. The `MAPPINGFUNCTION` describes a function to convert a value from source parameter type to target parameter type. The following example defines a mapping from a '`ChangeState`' operation to a '`Dimm`' operation, where the '`State`' parameter of '`ChangeState`' operation is mapped to the '`State`' parameter of '`Dimm`' operation. Thereby is mapped the values '`on`' and '`off`' to '`100`' and '`0`'.

```
<MAPPINGS>
  <MAPPING>
   <OPERATIONLINK>
    <CONTEXT>/Home/Conditions/Light/LIGHT/v1.0</CONTEXT>
    <OPERATIONNAME>ChangeState</OPERATIONNAME>
   </OPERATIONLINK>
   <TARGETOPERATIONNAME>Dimm</TARGETOPERATIONNAME>
   <MAPPINGINPARAMETER>
    <SOURCEPARAMETERNAME>State</SOURCEPARAMETERNAME>
    <TARGETPARAMETERNAME>DimmValue</TARGETPARAMETERNAME>
    <MAPPINGFUNCTION>{("off", "on"),("0", "100")}</MAPPINGFUNCTION>
   </MAPPINGINPARAMETER>
  </MAPPING>
</MAPPINGS>
```

4.3.2.5 Descriptions

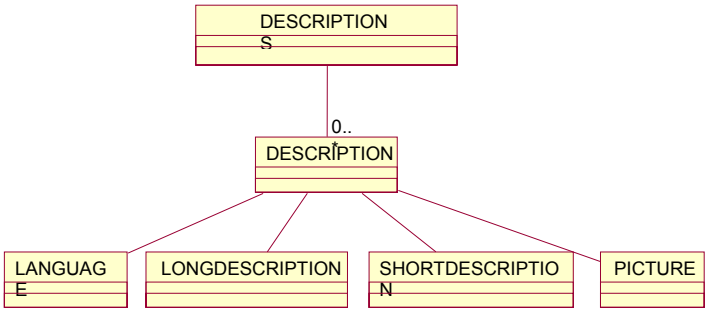


Figure 74: Ontology Structure: DESCRIPTIONS

The DESCRIPTION data is available in different places inside ontology to describe a (sub)structure respectively any field. The structure DESCRIPTIONS can contain more than one DESCRIPTION structures, one for each supported language. The DESCRIPTION structure consists of LANGUAGE field, LONGDESCRIPTION field, SHORTDESCRIPTION field, and PICTURE field. The field LANGUAGE defined the supported language. The fields LONGDESCRIPTION and SHORTDESCRIPTION contain descriptions in a long and a short version respectively. The different versions serve different purposes. The shorter one is better to present in speech-based user interfaces, but the longer one is probably better to understand. The field PICTURE contains a URL, which refers to a picture that should help illustration the provided functionality.

```
<DESCRIPTIONS>
  <DESCRIPTION>
   <LANGUAGE>EN</LANGUAGE>
   <LANGDESCRIPTION>
    a dimmer can change the light intensity in a smooth manner
   </LANGDESCRIPTION>
   <SHORTDESCRIPTION>Dimmer</SHORTDESCRIPTION>
   <PICTURE>http://localhost:8080/Images/SDO/DIMMER.gif</PICTURE>
  </DESCRIPTION>
</DESCRIPTIONS>
```

4.3.3 Ontology Server Maintenance Tool

The Ontology Server Manager offers all functions to manage and to discover the content of the Ontology Server. The graphical user interface is segmented into three parts, namely ‘General’, ‘Manage’, and ‘Request’. ‘Manage’ shows all registered ontologies into a tree-structure. The branches of this tree reflect the CONTEXT, ONTOLOGYNAME, and VERSIONNUMBER parts of registered ontologies. Figure 75 shows an ontology with context SAMPLE/DEVICE/ACTIVEDEVICE. The ONTOLOGYNAME is ‘LIGHT’ and the ‘VERSIONNUMBER’ is ‘1.0’. This part offers the functions ‘update’ and ‘insert from file’. The command ‘update’ reads the current content of Ontology Server and flushes the information of Ontology Server Manager. The command ‘insert from file’ enables the registration ontology from a given file.

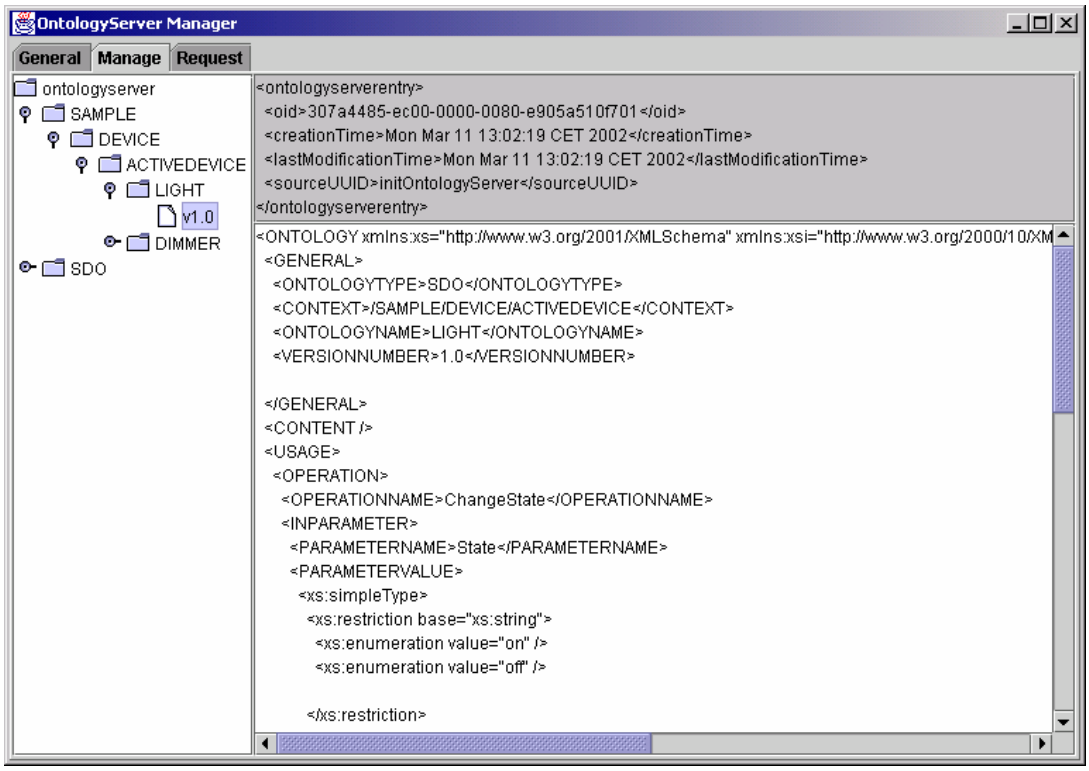


Figure 75: Ontology Server Manager

'General' shows the OID, time of creation, time of last modification, and the UUID of the instance, which has registered respectively last modified this ontology. 'Request' is for testing purpose only. It is possible to issue preconfigured requests to the ontology server to check its behavior.

4.4 I-centric Service

I-centric Service components offer ambient-aware, personalized, and adaptive behavior to individuals (see section 2.2.3). The actual functionality of such services depends on the application domain. However, an I-centric Service component consists of a higher-level part, which comprises the application domain specific service logic and a lower-level part that deals with communication aspects inside the framework.

The same requirements apply for the lower-level communication part as for the Context Interpreter: The I-centric Service has to be aware of changes in profiles of interest. Therefore, it can subscribe with the Context Server as Context Interpreters do and must then offer an interface, to receive resulting notifications.

I-centric Services process the information using their service logic, which results in specific actions feasible for this special situation. The appropriate SDOs for these actions are selected and activated. By use of the *Profile Generator*, the I-centric Service creates profile information about the performed actions and sends it to the Context Server. This kind of information can for instance be used for automatic customization of services and enhancement of human-machine interactions.

4.4.1 I-centric Service Interface

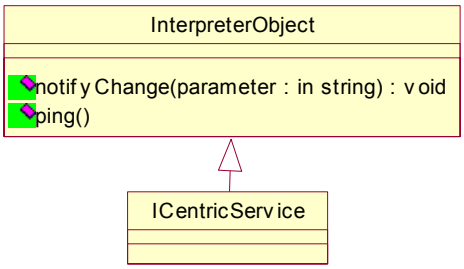


Figure 76: Interface of an I-centric Service

Since an I-centric Service's functionality is equivalent to the one of the Context Interpreter, the usage interface is equal, too. It offers only one operation for the notification of changes in subscribed profile information.

I-centric Service interface	
Operation	Description
`notifyChange`	Notification that the subscribed data has been changed in a profile. The changed data structures are given as parameter. Note: Needs previous subscription at the Context Server.
`ping`	Verify if the component is up and running

Table 6: Operations of the I-centric Service interface

The operation is called with one XML parameter. Please check `notifyChange` description above in section 4.2.1.2.

In contrast to the Context Interpreter, an I-centric Service communicates not only with the Context Server but also with selected SDO components. The higher-level part of the I-centric Service, which holds the actual service logic, processes the received profile information and may come to the decision that certain actions have to be performed. Since only SDOs have access to the underlying components of the Sensing and Controlling Environment, the I-centric Service communicates with these components, too.

4.4.2 Profile Evaluation and Service Logic

The evaluation of profiles inside the proposed framework is an important. Profile Evaluation takes place in Context Interpreters and I-centric Services for reasons of decision-making, service provisioning, and profile information evaluation.

The eXtensible Stylesheet Language Transformation [W-XSLT][31] has been chosen as enabling technology for Profile Evaluation, because it offers a standardized, open, and XML based methodology for expressing service logic. A component that evaluates profiles provides a XSLT interpreter that is able to process input XML data with use of algorithms that are externally specified using the XSL transformation language. The result can be XML typed data or just text formatted data.

The Profile Evaluation process (shown as flowchart in Figure 77) takes place in a Context Interpreter and I-centric Service. The process starts, when such a component receives new profile information as notification from the Context Server. Depending on its internal service logic or interpreter logic, the component requests additional profile information from the Context Server. The profile information from the notification and the data from the additional request are merged to one piece of XML data. The resulting XML tree contains the requested data with its original structure and the notification data as part of this structure.

[31] For more information about XSLT, see section 7.1.8.

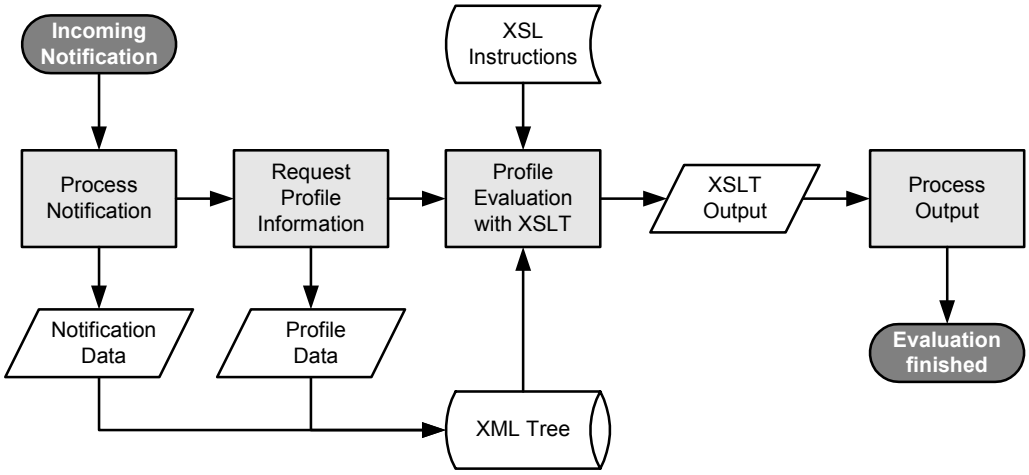


Figure 77: Profile Evaluation

The XML tree is used as input for the XSLT interpreter. XSL instructions that specify the actual service or interpreter logic are provided as input for the interpreter too. The result of the XSLT Profile Evaluation is XML data that is parsed by the component. Depending on the type of the component, the content of the resulting XML data can vary. A Profile Evaluation process of a Context Interpreter produces XML data that contains new profile information, which is sent back to the Context Server (i.e. an augmented profile). An I-centric Service produces XML data that contains instructions about actions, which have to be performed on certain Super Distributed Objects.

The actual component logic is specified in the XSLT transformation language. The logic obviously depends on the intended purpose of the Context Interpreter or I-centric Service. Therefore, no further description can be given here on the topic of algorithm specification.

The advantage of this Profile Evaluation methodology results from the fact, that the processing logic of framework components is externally specified in an XML compliant notation. This enables the provision of flexible and modifiable algorithms as described in section 3.2.6.4.

4.4.3 Request Control

Request control is responsible for concurrent access to physical resources. Like an I-Centric service, it subscribes for specific profile information inside the Context Server. However, the only information, the Request Control is interested in, is ‘request information’. An I-centric Service can issue a request that it wants to control a certain SDO. The Request Control will be notified about such requests. It resolves the participation SDOs, and controls them to fulfill the service.

If concurrent requests occur, the Request Control can queue or block these requests. E.g., a user is inside a room and has requested the light to be switched ‘on’. A second user enters later on, which wants to have to light switched ‘off’. The request of the second user will be queued, because of the conflict to the request of the first user. If the first user leaves the room, the request of the second user will be executed.

This is a first approach for managing concurrent access. Future research should investigate how to apply permissions, prohibitions, and roles to the request control. A possible scenario can be: A user, that is visiting, enters a room and switches the light ‘on’. After that, the owner of the apartment enters the room and wants the light to be switched ‘off’. If the Request Control is aware of the relationship between these two persons (owner - guest) the light can immediately switched ‘off’.

4.4.4 Context Interpreter

Context Interpreters subscribe a set of categories at the Context Server (Context Profile or Relation Profile). They will be notified if information concerning these categories has changed.

Context Interpreters processes the notifications of the subscribed data from the Context Server and generate information feasible for I-centric Service components. Therefore, Context Interpreters need some kind of knowledge about the syntax and the semantic of the profile information provided by SDOs. They are able to process this information using a *Profile Analyzer*. Dependent on the received profile information they can generate metadata (i.e. higher-level representations of the contextual information) or other new profile information, which is feasible for I-centric Services (e.g. a new relationship between SDOs). In general, a Context Interpreter analyzes information sent by SDOs, generates new data, and sends it back to the Context Server.

Example: If an individual is detected by any sensing technology, a technical ID will represent him in an SDO profile, which is stored in the Context Profile. A Context Interpreter can for instance generate a relationship between the individual and the room. This relationship is sent back to the Context Server and stored in the Relation Profile.

A single Context Interpreter does not necessarily perform the complete processing of data sent by an SDO. It is also possible to have a multistage processing by several Context Interpreters, which results in profile information with different abstraction levels.

In general, a Context Interpreter is responsible for the processing of profile information. The actual purpose of a Context Interpreter depends on the application domain it has been created for. For instance, a Context Interpreter can be responsible for the mapping of concrete profile information to an abstract representation and vice versa (i.e. that a temperature of 25°C has the meaning '*warm'* for a certain individual). Alternatively, it can manage the creation and modification of relationships between certain profile parts (i.e. the relation between an actor and a room, if the actor has been located in this room by some kind of sensor device).

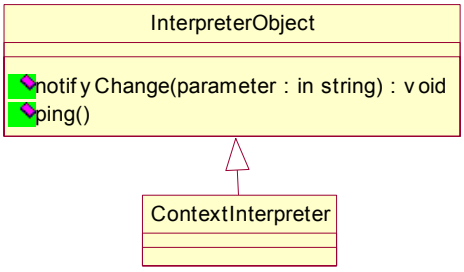


Figure 78: Interface of a Context Interpreter

For the provision of such functionality, a Context Interpreter processes new or changed data inside the profile it is responsible for. Therefore, it has to 'know', when the data has changed. This can be realized either if it uses some kind of polling mechanism to retrieve the profile information every time a fixed amount of time has elapsed, or it subscribes at the Context Server for this data to get notified about changes. For the latter method, a Context Interpreter must offer an interface, where it can receive such notifications.

The interface of the Context Interpreter provides two operations `notifyChange`, and `ping`. As the Context Interpreter is derived from the same parent than the I-centric Service (InterpreterObject), the descriptions of the two functions can be found in section 4.2.1.

4.4.5 Service Builder

The Service Builder is a tool that implements the mechanisms necessary for the interactive creation of I-centric Services as described in section 3.2.7.1. Figure 79 shows the implemented graphical user interface of the Service Builder. The application guides a user trough the service creation process by asking for service triggers, conditions, and actions. Conditions are combined into rules. One trigger can cause the evaluation of several rules. One service can have several triggers.

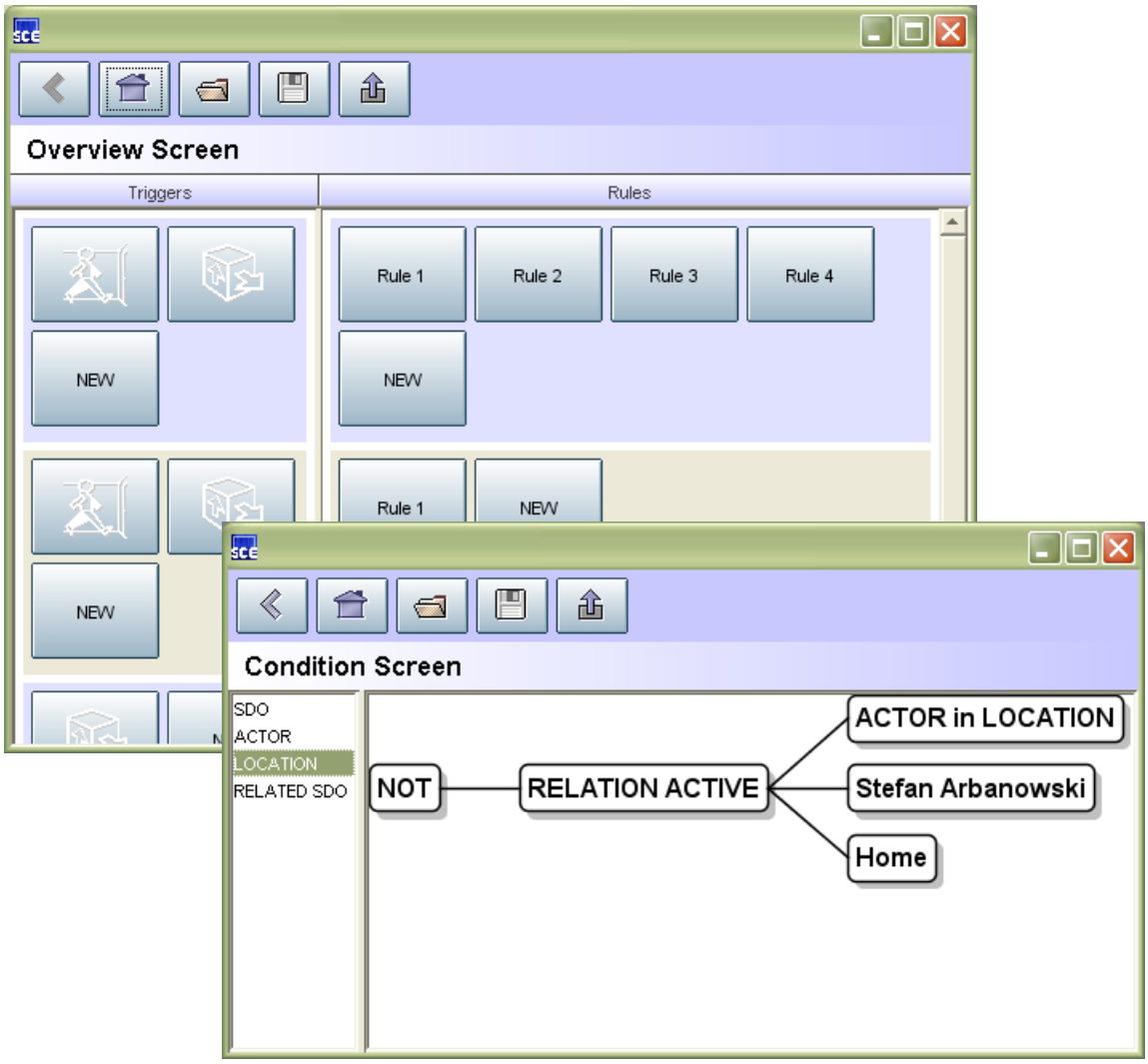


Figure 79: Service Builder GUI

Figure 79 also shows one example of a condition. In this case, the I-centric Service has to check whether a certain actor (Stefan Arbanowski) is at a certain location (Home). This is expressed as a relation (type of relation is 'Actor in Location') between an actor and a location.

After a user has finalized the configuration of the new service, the user can deploy and start the service by one click. This 'one click' initiates the creation of an XML document that contains the configuration of the new I-centric Service, and an XSL document that contains the business logic of the new I-centric Service. Examples of XML and XSL documents, which are generated by the Service Builder, are given in section 7.4.1 (XML) and 7.4.2 (XSL). These documents are deployed in the I-centric communications system and a new instance of an I-centric Service is started, which is configured to use the new generated files as input. Furthermore, a user can manage all services he has initiated in terms of starting, stopping, modifying, and deleting them.

4.5 Super Distributed Objects

Super Distributed Objects (SDOs) represent the physical objects and individuals of the real world as well as abstract and non-physical concerns of an individual's communication space. They can be seen as virtual counterparts of real world entities.

The SDO specification introduced in the following is two-fold. The first part represents the SDO usage interface provided towards the Open Profiling Framework of I-centric communications systems. As introduced in section 3.3.4, the usage interface is provided by the 'SDO specific' interface implementation. The second part of the SDO specification deals with the non-functional interfaces needed for management and platform integration issues. These interfaces enable the reservation, configuration, and monitoring of Super Distributed Objects.

4.5.1 SDO Usage Interface

SDOs form a connection to the underlying *Sensing and Controlling Environment*. They communicate their state to the Context Server. They gather information about physical resources and individuals they represent using the sensing and controlling environment. Ambient information is sent to the Context Server where it is further processed and stored in profiles. All gathered information is expressed in a representation that is feasible for the communication with the Context Server. In addition, they can interact with the controlling environment to have actions performed in the real world.

For this purpose, an SDO offers a usage interface via which it can be controlled.

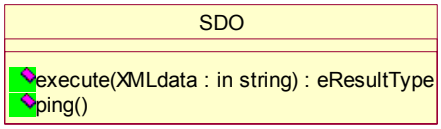


Figure 80: Usage Interface of a Super Distributed Object

An SDO can for instance represent the lighting in a certain room. In this case, the SDO would gather information about the current state of all light devices in the room. Changes of this state (e.g. someone turns a light device on) are sent to the Context Server. Additionally, the SDO can be instructed to alter the state of a light device if needed for an I-centric Service.

Super Distributed Object interface	
Operation	Description
`execute`	Requests the execution of an action on a real world artifact. The action is described by the parameter of the operation.
`ping`	To verify if the component is up and running.

Table 7: Operations of the Super Distributed Object Usage Interface

The SDO usage interface comprises only one operation, which is used to request certain actions on real world objects. The intended action is described in the parameter of the operation. Because there is a multitude of potential devices, the operation can only form a generic placeholder for an action request.

```
<EXECUTE>
 <OPERATION NAME="ChangeState">
  <PARAMETER NAME="State" VALUE="off"/>
 </OPERATION>
</EXECUTE>
```

The appropriate syntax and the semantic belonging to a certain request are completely SDO-specific and depend on the application domain of the actual component. The actions that can be performed by a specific SDO have to be described in the corresponding ontology, which is stored in the Ontology-Server.

Each SDO checks incoming execute commands against its own ontology (by asking the Ontology Server) to prevent the execution of invalid commands. Except this formal checking, the interpretation, evaluation, and mapping to the SDO specific representations of the execute

command has to be provided by the SDO programmer. Only the programmer has the knowledge how to map and to evaluate. Therefore, this kind of logic has to be embedded in each SDO separately.

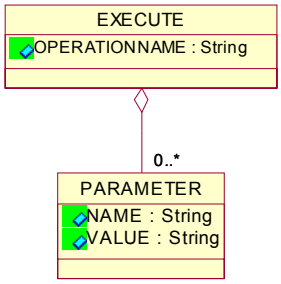


Figure 81: Parameter of the `execute` operation

The parameter for the `execute` operation consists of an `EXECUTE` element with the name of the requested operation and an optional list of parameters. Each of such parameter holds a name/value pair. There is no exact semantic attached to this model. The semantic of the parameter is defined in on ontology and has to be registered within the Ontology Server.

If information inside an SDO occurs that has to be notified to the Open Profiling Framework, an SDO uses the `notify()` method of the Context Server. Examples for `notify` and `execute` messages are given together with the description and specification of the implemented SDOs in section 4.7.3.

4.5.2 SDO non-functional Interfaces

This section describes a Platform Independent Model (PIM)[32] for resource data and interfaces of Super Distributed Objects. The model depicts the specification of SDO functions and resource data with regard to the standardization work undertaken by the OMG SDO DSIG[33].

First, the general resource data model is introduced. Based on this, the proposed SDO interfaces are presented in detail, including:

- the *SDO* interface,
- the *SDOService* interface,
- the *Discovery* interface,
- the *Monitoring* interface,
- the *Configuration* and *ConfigurationExt* interfaces, and
- the *Reservation* interface.

Note: The SDOService interface, introduced in the following, has been used during the practical implementation to realize the SDO Usage interface as described in section 4.5.1. In fact, the SDOService interface is not a non-functional interface. But, the actual operations of the SDOService interface have to be added by domain specific implementations, as they are not part of the SDOService interface specification. In this sense, it is non-functional. That reflects exactly the intention of the SDOService interface, to be used for domain specific interface implementation (in this case the Usage interface of I-centric objects).

[32] The Model Driven Architecture (MDA) is a new way of writing specifications and developing applications, based on a platform-independent model (PIM). A complete MDA specification consists of a definitive platform-independent base UML model, plus one or more platform-specific models (PSM) and interface definition sets, each describing how the base model is implemented on a different middleware platform. [MDA]

[33] Object Management Group (OMG) Domain Special Interest Group (DSIG) for Super Distributed Objects (OMG DSIG SDO)

The model as presented in this document is a platform independent model. Therefore, it must not use any platform specific descriptions or technologies. However, for simplifying the work and the testing, this model uses some CORBA specific data types, such as *any* and *sequences*. These data types however can be adapted easily to standard UML data types if needed. Nevertheless, any other platform specific model that is not based on CORBA can be derived from the presented PIM.

4.5.2.1 Resource Data Model

This section describes the resource data model. The model comprises classes and data types that are used to model capabilities and properties of Super Distributed Objects. The class diagram of the detailed resource data model is shown in Figure 82.

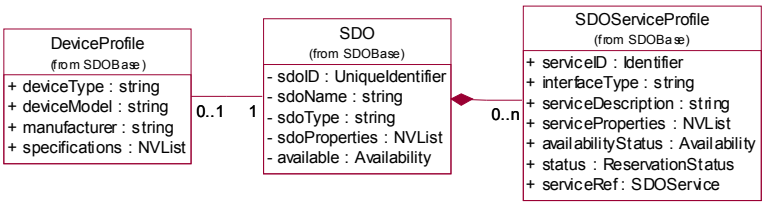


Figure 82: Resource Data Model

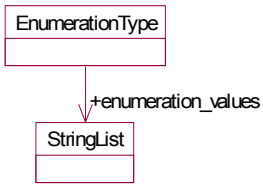
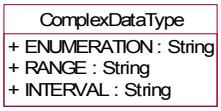
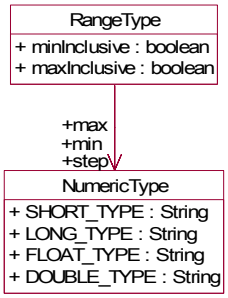
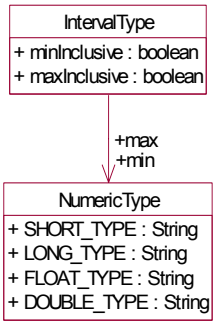
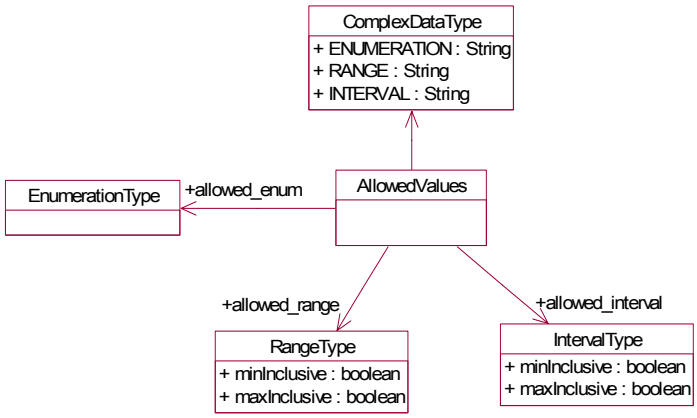
According to the model given in Figure 82, the core part of an SDO is represented by the SDO class. If the SDO has an underlying device, this device is represented by the class *DeviceProfile*. An SDO can have only one *DeviceProfile* element. Please note, that an SDO can provide one or more services. However, SDOs representing one or more services or software components do not have a device profile. Each of these services is represented by an *SDOServiceProfile*, which describes the services' functionalities and capabilities. These classes are discussed in detail below.

General Data Types

This section describes the general data structures used in the SDO specification.

Name	Description
Identifier	Identifier for the objects in the resource data model (e.g. SDO). Each identifier is typed as a string. The uniqueness of the identifier values depends on the object it is identifying. If it is identifying SDOs then it must be unique in a given domain of application deployment.[34] If it is identifying a stored configuration in an SDO then it may not be unique in a given domain of application deployment.
StringList	Data type describing array of *strings*.
NameValue NameValue + name : string + value : any	A pair of a name and its value. A name value pair used to store all properties or attributes of an SDO. - name - The name of the attribute whose value is being stored. - value – The value of the attribute. Please note that the type of the attribute is described by the data type 'Parameter' which is described later on.
NVList	This data type defines a sequence of type 'NameValue'.
NumericType NumericType + SHORT_TYPE : String + LONG_TYPE : String + FLOAT_TYPE : String + DOUBLE_TYPE : String	The enumeration of numerical types is supported. The supported types are *short, long, float,* and *double*.

[34] The implemented I-centric Communications system uses a standardized scheme, IETF UUID [UUID00], to create unique identifies.

Name	Description
EnumerationType 	Data structure representing the possible values that can be assigned to an attribute defined as enumeration. The enumerated values are always of *string* type. EnumerationType is used when an attributes may contain only certain values. For example, an airconditioner SDO which has three different speed levels (*high*, *medium,* and *low*) can have its attribute defined as *speed{"high", "medium", "low"}*. This will ensure that only a valid value for the attribute is given.
ComplexDataType 	Data structure representing complex data types are supported. The allowed complex types are *enumeration*, *range*, and *interval*. These complex types are defined above.
RangeType 	Data structure representing the values that can be assigned to a 'Parameter' defined as range (please see below where data structure 'Parameter' is defined). - *min* – the lower bound of the range - *max* – the upper bound of the range - *minInclusive* – a boolean value showing if the lower bound value is included in the range - *maxInclusive* – a boolean value showing if the upper bound value is included in the range - *step* – the step between the values in the range. Example: the range ((int) 0, (int) 15, false, true, (int) 5) allows values {5, 10, 15}.
IntervalType 	Data structure representing the values that can be assigned to a 'Parameter' defined as interval. - *min* – the lower bound of the interval - *max* – the upper bound of the interval - *minInclusive* – a boolean value showing if the lower bound value is included in the range - *maxInclusive* – a boolean value showing if the upper bound value is included in the range Example: the interval ((int) 0, (int) 20, false, true) allows values {1,2,3,…,19,20}.
AllowedValues	 Data structure representing the values that a 'Parameter' can contain if it is defined as a complex type. If it is defined as enumeration, range and interval, a set of allowed values is specified with *EnumerationType*, *RangeType* and *IntervalType*, respectively.
Parameter	Data structure to define an attribute ('Parameter') independently of implementation technologies. The *Parameter* structure defines the name of variable and the type of data it may contain. The type of the attribute can be simple types defined by the implementation technology or it can be complex type specified for that purpose.

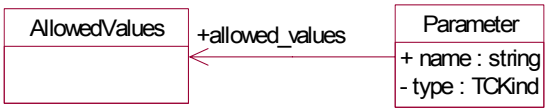
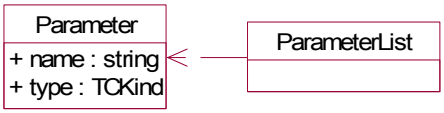
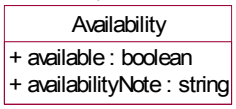
Name	Description
	 Please note that the 'Parameter' defines the type of the value of an attribute only. Its actual value is stored in the 'NameValue' data structure. The name in both 'Parameter' and 'NameValue' must be the same so that the value of an attribute can be found without any problem. - name – The name of the attribute. - allowed_values – The type of the attribute.
ParameterList	 The datatype describing an array of *Parameters*.
Availability 	Data type defining the current availability of a device or a service an SDO is representing. *available* – Flag indicating if the object is available or not. - *true* – available - *false* – not available - *availabilityNote* - Comments regarding *availability* especially if it is not available.

Table 8: General Data Types

Exceptions in SDO

The SDO specification defines a number of exceptions (e.g. `SDO_NOT_FOUND`, `INTERFACE_NOT_IMPLEMENTED`, `SERVICE_NOT_FOUND`, `INVALID_ARGUMENT`, `OPERATION_FAILED`). These exceptions are raised to notify error states inside an SDO based system. A complete list of all exceptions defined can be found in [SDO-Sub]. The IDL specification of the SDO exceptions is also given in section 7.5.3.1.

4.5.2.2 Core SDO Class

Super Distributed Objects are represented by instances of the class *SDO*. They implement the *Discovery* interface and can optionally implement the *Monitoring*, *Configuration* (or *ConfigurationExt*), or the *Reservation* interface.

Figure 83: Class SDO

Attributes of the SDO Class

Attribute	Type	Description
sdoID	Identifier	The SDO identifier. The value of this attribute must be a globally unique value. Furthermore, the value of this attribute is constant for the life time of the SDO.
sdoName	string	The human-readable name that can be used to present the SDO for user (in an end-user application).
sdoType	string	The type of the SDO indicating its functionality. It may have values like LightSwitch or TV Set. If the SDO wraps a device specified by the class *DeviceProfile*, the value of type attribute corresponds to the value of deviceType attribute of this class.
sdoProperties	NVList	The list of parameters specifying SDO properties. For each parameter, the list contains its name and current value.

Attribute	Type	Description
available	Availability	The attribute indicating if the underlying device or software component of the SDO is currently available. In addition, it contains the possible reason of device unavailability, for example network failure or some device-specific problems, as text.

Table 9: Attributes of Class SDO

Class DeviceProfile

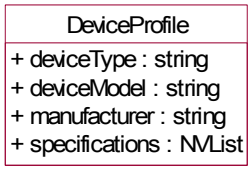


Figure 84: Device Profile

The class *DeviceProfile* describes the device wrapped by the SDO. The existence of a device profile is not mandatory; there can be SDOs that offer only services without any physical device.

Nevertheless, an SDO can only represent one physical device. Therefore, it provides at maximum one instance of the *DeviceProfile*. The attributes of this class reflect the device characteristics.

Attributes of the DeviceProfile Class

Attribute	Type	Description
deviceType	String	The type of device in string representation, for example *printer* or *light switch*.
deviceModel	String	The name of device model given by manufacturer.
manufacturer	String	The name of device manufacturer.
specifications	NVList	The list of static device characteristics and their values. The content of the list depends on the device type and model. For a computer screen properties may comprise the range of possible screen resolutions and supported frequencies.

Table 10: Attributes of Class DeviceProfile

Class SDOServiceProfile

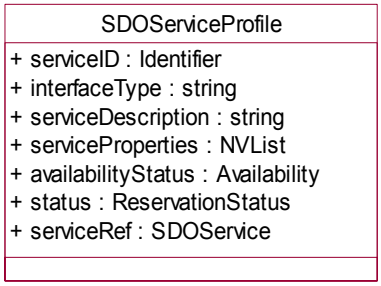


Figure 85: SDOServiceProfile Class

An object of the class *SDOServiceProfile* represents services that any SDO can provide. Corresponding to the number of services, the SDO class can contain zero, one or more instances of the class *SDOServiceProfile*. This class describes all information of the service, including its functionalities, availability, and the reference to the service object.

Attributes of the SDOService Profile Class

Attribute	Type	Description
serviceID	Identifier	The unique identifier of the service; its value must be unique across all services of this SDO. Please note that this identifier must not be globally unique.

Attribute	Type	Description
serviceProperties	NVList	The list of parameters specifying service attributes. For each parameter, the name and current value are contained in the list.
interfaceType	String	The type of the service interface. The attribute value refers to interface specification in an interface repository. The specific format of this value and kind of repository depends on the technology supported by the SDO. For SDOs based on CORBA this may be the IDL interface type, for instance, "IDL:LightDevices/Switch:1.0". The attribute remains constant value for the SDO life time.
serviceDescription	String	A brief human-readable description of the service semantics, for example, *switch* or *sendMessage*.
availabilityStatus	Availability	This attribute indicates whether the service is currently available. In addition, it contains the possible reason of unavailability, for example *network failure*, as text.
status	Reservation-Status	The reservation status of the service. This status description defines if and how the service is reserved by another SDO. Other SDOs can use this attribute before reserving or executing the service.
serviceRef	Service	The reference to service object represented by this service profile.

Table 11: Attributes of Class ServiceProfile

Resource Data Model Examples

To cover the overall concept of the Super Distributed Objects (SDO) three different types of SDOs (*Thermometer* SDO, *Airconditioner* SDO, and *TemperatureController* SDO) are described as example SDOs. These three examples are used throughout this document to illustrate the interfaces or their usages.

Thermometer SDO is a simple SDO representing a physical device, a thermometer that senses the temperature of a room. It does not provide any service and has only two monitoring parameters. The *Airconditioner* SDO is a more complex SDO with an air-conditioner as the underlying device providing two services, cooling and heating. *TemperatureController* SDO is an SDO that does not represent any physical device but provides a service that controls the temperature of the room.

The following examples will describe the resource data model of SDOs with concrete values.

DeviceProfile		
Attribute	Value	
deviceType	*TemperatureSensor*	
deviceModel	*TH310*	
Manufacturer	*Thermo Inc.*	
specifications	name	value
	rangeMin *rangeMax*	*-50* *150*

SDO		
Attribute	Value	
sdoID	*abc_2234_5567*	
sdoName	*ThermoSensor*	
sdoType	*TemperatureSensor*	
sdoProperties	name	value
	location *value* *unit*	*Meeting_room* *25* *Celcius*
available	available *true*	availabilityNote

Figure 86: Thermometer SDO

Thermometer SDO is a simple SDO representing a physical device whose profile is described by the *DeviceProfile*. It does not provide any service. Therefore, it does not implement *SDOServiceProfile* class. One of the parameters of the *Thermometer* SDO named *value* holds the current temperature of the room wherever it is located (*Meeting_room*).

This value can be monitored by other SDOs but it cannot be changed or configured. Another parameter *unit* denotes the unit (degrees Celsius or degree Fahrenheit) of *value*. This parameter can be monitored and configured.

SDO		
Attribute	**Value**	
sdoID	*mno_8876_0987*	
sdoName	*temperatureController*	
sdoType	*temperatureService*	
sdoProperties	name	value
	location	*Meeting_ room*
available	available *true*	Availability Note

SDOServiceProfile		
Attribute	**Value**	
serviceID	*tempContrl_112233*	
interfaceType	*temperatureController*	
serviceDescription	*Keeps the temperature of the room within certain level.*	
serviceProperties	Name	value
	minTemp	*22*
	maxTemp	*25*
availableStatus	available	availabilityNote
	true	
status	mode	Available After
	FREE	
serviceRef	*Temp controller service reference*	

Figure 87: TemperatureController SDO

The `TemperatureController SDO` is a service SDO that does not represent any physical device. Its function is to keep the temperature of a room at some predefined level. When the temperature of the room is below the predefined level `minTemp` it turns the heater of the air-conditioner `on`, if the temperature of the room goes above `maxTemp` it turns the cooler `on` and if the temperature is between these two predefined levels it turns the air-conditioner `off`. These two properties of the service `minTemp` and `maxTemp` can be configured.

This SDO needs both SDOs, the `Thermometer SDO` and the `Airconditioner SDO`, to work properly. It monitors the temperature of the room provided by the `Thermometer SDO` and can then invokes either one of the two services provided by the `Airconditioner SDO`.

4.5.2.3 SDO Interface

In addition to its usage interfaces, an SDO shall offer operations intended for its resource management. Whereas the usage interface is specific to individual SDO types, the management interfaces are common to all SDOs. However, because some of the management interfaces are optional, not every SDO has to provide them.

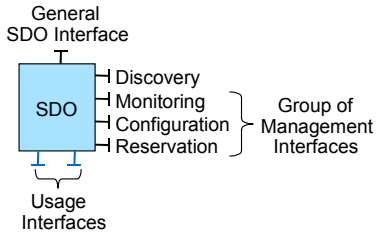


Figure 88: Interfaces in Detail

The operations of the management interfaces can be divided further into *Discovery*, *Monitoring*, *Configuration*, and *Reservation* as depicted in Figure 88. The *Discovery* interface enables an SDO to advertise its capabilities in the SDO network and likewise to find services it requires.

The *Monitoring* interface enables to watch the current state of SDO resource data. The SDO can be configured through the operations of the *Configuration* interface; moreover, the current configuration data can be accessed as well.

The *Reservation* interface permits to specify certain SDO utilization schemes. The general *SDO* interface provides the initial access to an SDO including access to the other interfaces.

The general *SDO* interface as well as the *Discovery*, *Monitoring*, *Configuration*, and *Reservation* interfaces is described in detail in the following sections.

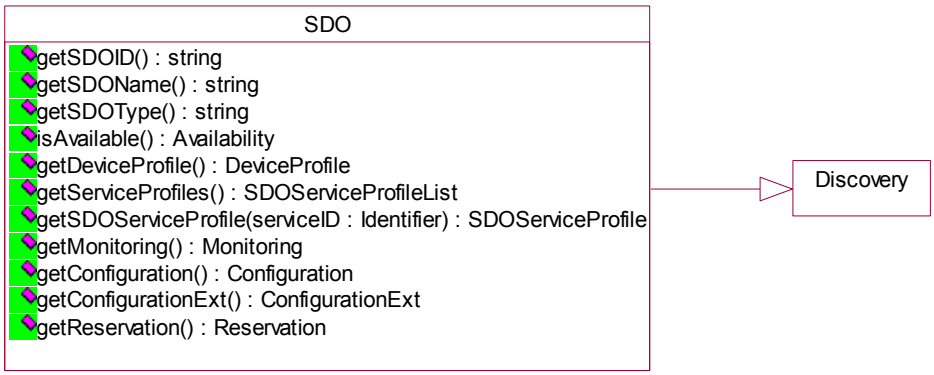


Figure 89: SDO Interface

Data Structures defined for the SDO Interface

Name	Description
NVList	The data type defines an array of objects of type 'NameValue'.
Availability	Data type defining the current availability of a *device* or a *service* an SDO is representing.
DeviceProfile	An object corresponding to a physical device an SDO is representing.
SDOServiceProfile	An object representing services an SDO is providing.
SDOServiceProfileList	An array of 'SDOServiceProfiles'.
Monitoring	An object implementing the *Monitoring* interface.
Configuration	An object implementing the *Configuration* interface.
ConfigurationExt	An object implementing the *ConfigurationExt* interface.
Reservation	An object implementing the *Reservation* interface.

Operations provided by the SDO Interface

The operations in the interface of a Super Distributed Object provide access to the most common information about an SDO object, characterizing resources and functionality of the underlying device or software component, properties of SDO middleware component and description of services offered by the SDO.

Operation	getSDOID() : Identifier
Return Value	The Unique Identifier of the SDO.
Description	The operation gets the unique ID identifying an SDO

Operation	getSDOName() : string
Return Value	The human-readable name of the SDO.
Description	The operation returns the human-readable name that can be used to identify the SDO.

Operation	getSDOType() : string
Return Value	The type of the SDO.
Description	The operation returns the type of an SDO which describes its core functionality. It may have values like *TemperatureSensor* or *Airconditioner*. If the SDO wraps a device specified by the class *DeviceProfile*, the value of type attribute corresponds to the value of *deviceType* attribute of this class.

Operation	isAvailable(): Availability
Return Value	Availability of the SDO. *true*, if the SDO is currently available; *false* otherwise.
Description	The attribute indicates whether the SDO is currently available. In addition, it contains the possible reason of device unavailability, for example network failure or some device-specific problems, as text. The unavailability can be caused by device failure or local network problem.

Operation	getDeviceProfile() : DeviceProfile
Return Value	DeviceProfile of the underlying device if it exist.
Description	The operation returns the profile of the underlying the SDO represents a physical device. An appropriate error message is returned if the SDO does not represent a physical device.

Operation	getServiceProfiles() : SDOServiceProfile[]
Return Value	List of *ServiceProfiles* of all the services the SDO is providing.
Description	The operation returns *ServiceProfiles* of all the services it is providing.

Operation	getSDOServiceProfile(serviceID : Identifier) : SDOServiceProfile
Return Value	The profile of the specified service.
Description	The operation returns the specified service profile.

Operation	getMonitoring() : Monitoring
Return Value	The object representing the *Monitoring* interface of the SDO.
Description	The operation returns the object representing the Monitoring interface of the SDO. An appropriate error message is returned if the SDO does not provide the *Monitoring* interface. If an SDO has no *Monitoring* interface then none of its properties can be monitored.
Operation	getConfiguration() : Configuration
Return Value	The object representing the *Configuration* interface of the SDO.
Description	The operation returns the object representing the *Configuration* interface of the SDO. An appropriate error message is returned, if the SDO does not provide a *Configuration* interface. An SDO without *Configuration* interface means that it has no configurable properties.

Operation	getConfigurationExt() : ConfigurationExt
Return Value	The object representing the *ConfigurationExt* interface of the SDO.
Description	The operation returns the object representing the *ConfigurationExt* interface of the SDO. An appropriate error message is returned, if the SDO does not provide a *ConfigurationExt* interface. An SDO that does not offer the *ConfigurationExt* interface cannot store and activate predefined configurations.

Operation	getReservation() : Reservation
Return Value	The object representing the *Reservation* interface of the SDO.
Description	The operation returns the object representing the *Reservation* interface of the SDO. An appropriate error message is returned if the SDO does not provide a *Reservation* interface. An SDO without *Reservation* means that every SDO can invoke any services or configure its configuration parameters without restrictions.

4.5.2.4 SDOService Interface

Services of an SDO are represented by the class `SDOService`. This `SDOService` class can optionally implement the interface `Configuration` to configure the `service` parameters or the extended `ConfigurationExt` interface to allow saving different configurations of its properties as well as configuring them.

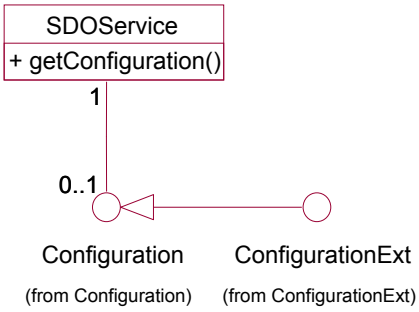


Figure 90: Class SDOService

The SDOService interface only represents a template to be extended for a certain application domain. An implementer of an SDO has to add new operations to the interface to provide the actual functionality of the implemented SDO to other components.

This has been done during the implementation of the Open Profiling Framework. The SDOService interface has been extended to provide the operations identified for the usage interface of I-centric objects.

Data Structures defined for the SDOService Interface

Name	Description
Configuration	An object implementing the `Configuration` interface.

Operations Provided by the SDOService Interface.

Operation	getConfiguration() : Configuration
Return Value	The object representing the Configuration interface of the SDOService.
Description	This operation returns the object representing the *Configuration* interface of the *SDOService*. An appropriate error message is returned if the *SDOService* does not provide a *Configuration* interface. An *SDOService* object that does provide *Configuration* interface either does not have any properties, which can be configured, or does not allow them to be configured.

4.5.2.5 Discovery Interface

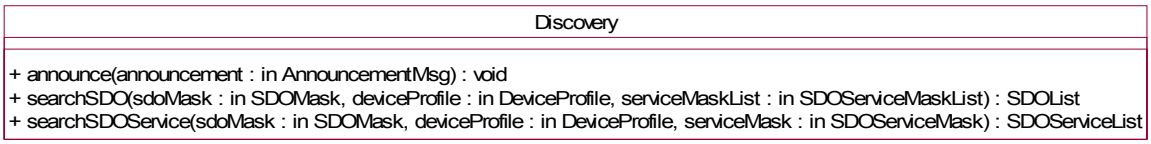


Figure 91: Discovery Interface

Data Structures defined for the Discovery Interface

Name	Description
AnnouncementType	enum AnnouncementType {NEW, ALIVE, LEAVE}; The enumeration AnnouncementType defines the types of announcement messages or in general describes the status of existence of the SDO sending the announcement in the network. NEW – Indicates the SDO is new in the network and this is its first message. ALIVE – Indicates the SDO has been in the network for sometime. LEAVE – Indicates the SDO is leaving the network.

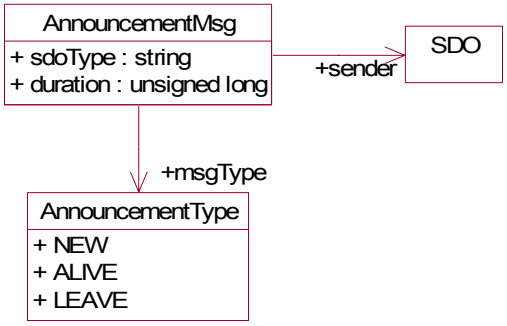
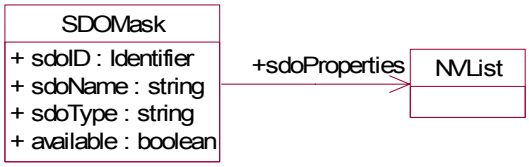
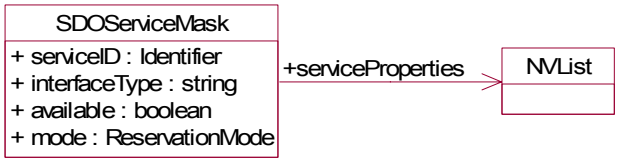
Name	Description
AnnouncementMsg	 AnnouncementMsg is the structure of the message sent when an SDO is sending an announcement. It describes the basic characteristics and functionalities of an SDO sending the announcement. - msgType – Type of the announcement message. Please see AnnouncementType described above. - sender – Reference to the SDO sending the announcement. - senderID – Unique identifier of the SDO sending the announcement. - sdoType – The type describing basic functionalities of the SDO sending the announcement - duration – Time duration (in milliseconds) until this message is valid. - serviceDescriptionList – List defining the services which are provided by the SDO.
SDOMask	 Defines basic SDO characteristics to facilitate the search process. When the search function of the *Discovery* interface is invoked with the mask as argument, any fields of the mask can be specified or left empty. Thereby, empty values match any input. The fields of 'SDOMask' correspond to attributes of the class *SDO*. E.g. if the only field specified in the mask is *sdoType*, and it is defined as Airconditioner, all SDOs of type Airconditioner are returned.
DeviceProfile	The class 'DeviceProfile', is used in search operations to define search masks for devices.
SDOServiceMask	 Defines basic 'SDOService' characteristics to facilitate search process. While invoking search function in *Discovery* interface any of these can be specified or can be left empty. Thereby empty values match any input. E.g. if only *interfaceType* is defined as *temperatureController* all services with the given type are returned.

Table 12: Discovery Interface Data Structures

Operations provided by the Discovery Interface

Operation	announce(announcement : AnnouncementMsg)
Return Value	non
Parameters	Announcement - The announcement message transmitted with this operation. The message defines the basic characteristics and functionalities of an SDO.
Description	This operation announces the SDO presence in the network. The announcement message is sent to all SDOs in the network in regular intervals so that the presence or absence of the SDO is known to other SDOs in the network.

Operation	SDOList searchSDO(sdoMask : SDOMask, deviceProfile : DeviceProfile, serviceMaskList : SDOServiceMaskList)
Return Value	List of SDO references matching the specified characteristics
Parameters	sdoMask - Defines basic SDO characteristics to facilitate search process deviceProfile - Profile of a device being searched serviceMaskList - List of characteristics defining services being searched
Description	This operation searches for SDOs in the network that have the specified characteristics. The characteristics are defined in sdoMask, deviceProfile and serviceMaskList. SDOs can be searched using any of these masks. Masks can be left empty, thereby matching any input.

Operation	ServiceList searchService(sdoMask : SDOMask, deviceProfile : DeviceProfile, serviceMask : SDOServiceMask);
Return Value	<return value> - List of Service references matching the specified characteristics
Parameters	sdoMask - Defines basic SDO characteristics to facilitate the search process deviceProfile - Profile of a device being searched serviceMask - List of characteristics defining services being searched
Description	This operation searches for services in the network with specific characteristics. The characteristics are constrained by sdoMask, deviceProfile and SDOserviceMaskList. Services can be searched using any of these masks. However not all the masks have to be specified - empty masks or empty attributes in a mask act like as wild cards.

4.5.2.6 Monitoring Interface

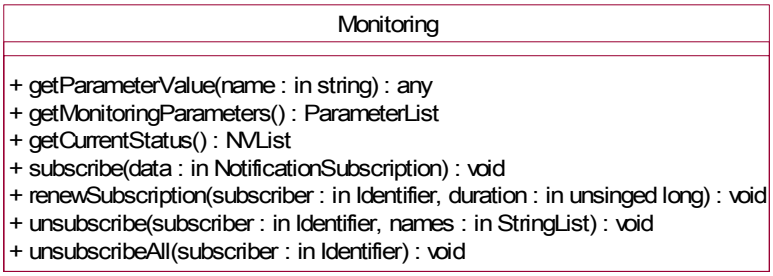
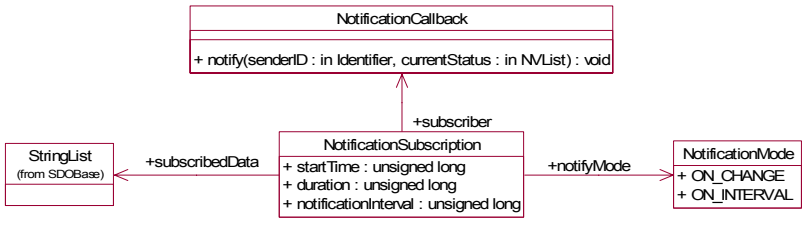


Figure 92: Monitoring Interface

Data Structures defined for the Monitoring Interface

Name	Description
NotificationMode	enum NotificationMode {OnChange, Interval}; Possible notification modes while subscribing to monitoring parameters. - ON_CHANGE - To get notification of the subscribed monitoring parameter every time the value of the parameter changes. - ON_INTERVAL - To get notification of the subscribed monitoring parameter on the specified interval of time. That is, if a subscription to some parameter is made in this mode, the notification is sent to the subscribing SDO only at the specified time with the current value.
NotificationSubscription	 This structure outlines the details of the subscription message. This message is received and evaluated by the SDO providing parameters for monitoring. Depending on the notification mode, the subscriber (the message sender) will receive appropriate messages containing the parameter value if the parameter changes periodically. - *subscriber* - The address or reference of the object that will receive notification messages; the object referenced here must implement the interface NotificationCallback. The value of this field depends on basis technology of the SDO system. - *subscriberID* – The unique identifier of the SDO that subscribes to this parameter.

Name	Description
	- *notifyMode* - The mode of notification (On Change or On Interval). The notifications are sent either when the value of at least one of subscribed monitoring parameters changes (notification on change), or periodically (attribute *notificationInterval*). - *subscribedData* - A list of monitoring parameters to be subscribed to. - *startTime* – Defines the time period (in milliseconds) at which monitoring of the parameters should start. If it is not specified, then the subscription will be activated right after receiving the subscription message. - *duration* - Indicates for how long (in milliseconds) the subscription should be last. - *notificationInterval* – Time interval (in milliseconds) in which the notification is sent to the subscribing SDO, if the *NotificationMode* of the subscription is *ON_INTERVAL*.

Table 13: Monitoring Interface Data Structures

Operations provided by the Monitoring Interface

Operation	getParameterValue (name : string) : any
Return Value	The current value of the parameter
Parameters	Name - Name of the parameter whose value is requested
Description	This operation returns the current value of the specified monitoring parameter

Operation	getMonitoringParameters () : ParameterList
Return Value	List containing names and types of monitoring parameters of the SDO
Description	This operation returns the list of monitoring parameters defined for this SDO.

Operation	getCurrentStatus () : NVList
Return Value	The list containing names and current values of all monitoring parameters of the SDO
Description	This operation returns the current values of all the monitoring parameters of the SDO.

Operation	subscribe (data : NotificationSubscription)
Parameters	Data - Parameters being subscribed and the conditions for the subscription
Description	This operation subscribes the calling SDO to the specified monitoring parameters. The details of the notification are denoted in the argument data. When a subscription request arrives, the SDO may add the subscriber to its internal table of subscribers. The subscriptions in the table can be distinguished by the identifier of the subscriber and the name of subscribed parameter.

Operation	renewSubscription (subscriber : Identifier, duration long)
Parameters	Subscriber – Unique ID of the SDO that is renewing the subscription Duration – Time duration until which the subscriptions of the specified SDO should be renewed
Description	This operation renews the already subscribed parameter for the specified duration of time. The subscription time is extended for all parameters that were subscribed previously by the specified SDO.
Operation	unsubscribe (subscriber : Identifier, names : string[])
Parameters	Subscriber - Unique ID of the SDO that is unsubscribing Names - List of names of the parameters being unsubscribed
Description	This operation unsubscribes the specified list of already subscribed monitoring parameters.

Operation	unsubscribeAll (subscriber : Identifier)
Parameters	Subscriber - Unique ID of the SDO that is unsubscribing all its subscriptions
Description	This operation unsubscribes all the subscribed parameters of the specified SDO.

Usage of the Monitoring Interface

Monitoring *ON_CHANGE* is useful if the subscribing SDO just wants to be notified as soon as one of the subscribed parameter value has changed. For example if the *TemperatureController* SDO subscribes temperature value in the *Thermometer* with *ON_CHANGE*, it receives notification about the changes in the temperature value every time the temperature value changes in the room.

The subscription of parameters is shown in the sequence diagram (Figure 93). To use the monitoring operations of the SDO *Monitoring* interface, a monitoring SDO (*sdo1* in the diagram) gets the object implementing the *Monitoring* interface of the monitored SDO (*sdo2*) first, as shown by message 1. Then, *sdo1* requests the list of monitoring parameters of *sdo2* (message 3). One or more parameters from the list are subsequently subscribed (message 5) to get notifications every time their values changes (notification on change).

After some time, when the values of one of these subscribed parameters change, *sdo1* gets a notification message with the name (or names) of the parameter that has changed along with its (or their) current values (messages 6, 7, 9). Shortly before the subscription expires, *sdo1* sends a renewal request to extend the subscription time (message 8).

If *sdo1* is no longer interested in some of the formerly subscribed parameters, it can unsubscribe them individually (message 10). After this, only the remaining parameters generate change notification messages (message 11). To stop getting notifications, *sdo1* unsubscribes all subscribed parameters (message 12).

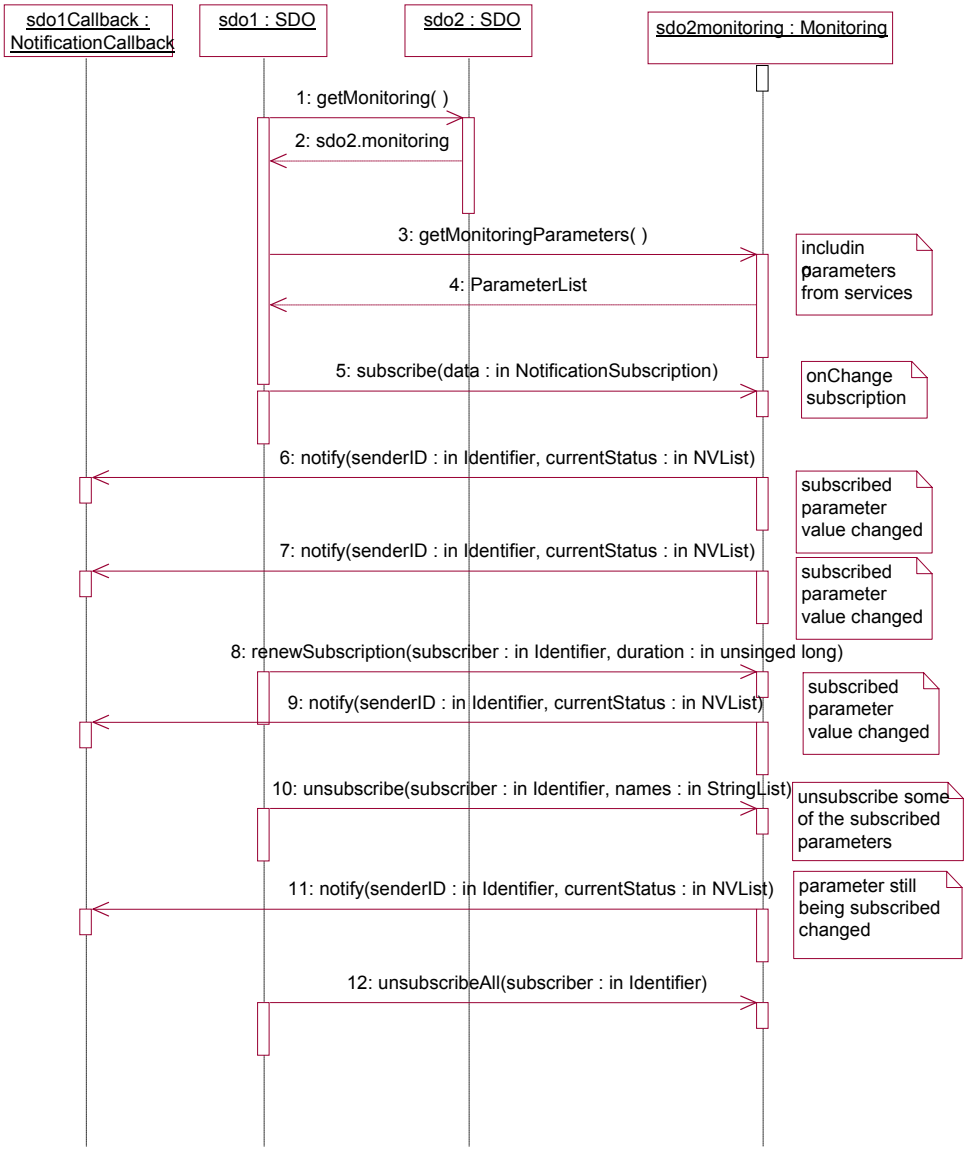


Figure 93: Subscription and Notification On Change

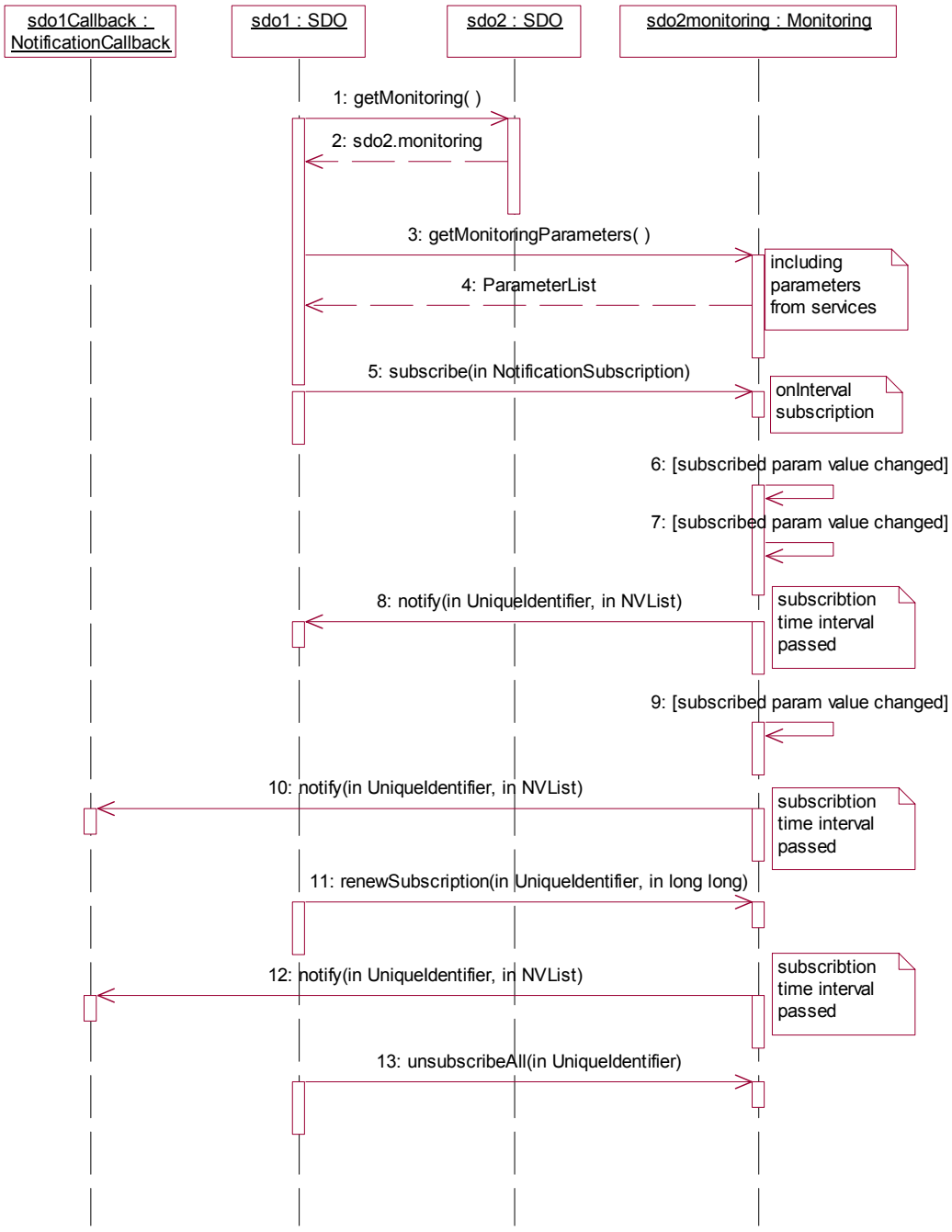


Figure 94: Subscription and Notification on Interval

Monitoring *ON_INTERVAL* is useful if the subscribing SDO wants it to be notified periodically, because it may want to observe the development of a specific parameter over a longer period. In the example of the *TemperatureController* SDO, subscribing temperature value, will receive the temperature of the room only in the specified time interval. Subscribing with this condition may have advantage against subscribing ON_CHANGE if the thermometer is very sensitive and senses very little difference in the temperature of the room.

Figure 94 shows the notifications made in interval mode. In general, the same sequence of operations shown in Figure 93 is used to subscribe, renew, and cancel parameter notification. The difference is that notifications are sent not in the event when one of subscribed parameters

changes its value, but periodically in a specified time interval. Notifications contain the parameter names and their values at the moment the notification was sent. Notifications are sent irrespectively of the fact whether the parameter values have changed or not since the last notification. Therefore, it can occur that between two notifications some parameters have changed their values more than once or never at all; for example, in Figure 94 the parameter values change twice between notification messages 5 and 8. Furthermore, it can also occur that parameter values remain unchanged during several notification intervals, like in the sequence diagram in Figure 94 between notification messages 10 and 12.

The `NotificationCallback` interface provides call back mechanism for an SDO for the subscription notification. Monitoring parameters of an SDO can be monitored by other SDOs by subscribing such parameters.

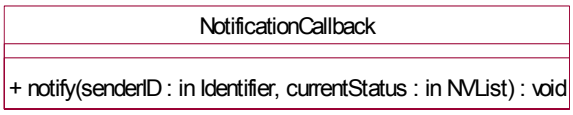


Figure 95: NotificationCallback Interface

The changes in the monitored parameters are subsequently propagated to the subscribing SDOs by notifications. This callback interface provides an operation to notify subscribing SDOs.

Operation	notify(senderID : Identifier, currentStatus : NVList)
Parameters	senderID - Unique ID of the SDO that is sending the notification currentStatus - A list containing the parameters and their current values. Please note that this list may not contain all the parameters that an SDO has been subscribed to, probably because not all parameter have changed or because the notification interval of some parameters has not elapsed yet
Description	Either this operation notifies the subscriber of a monitoring parameter that the value of the parameter has changed or the notification interval has elapsed.

The monitoring by polling consists in periodical checking the state of the SDO by the observer or checking the value of the property of an SDO manually. However, SDO parameters should be monitored primarily by the mechanisms described above.

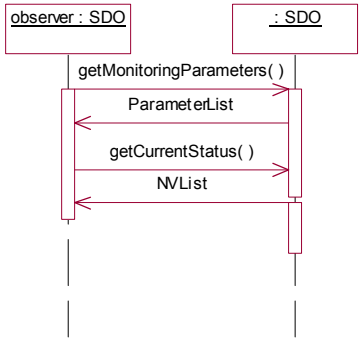


Figure 96: Message Sequence Chart: Monitoring by Polling

If an observer would like to trace the state of a certain SDO, at first it should retrieve information about the parameters that specify it. It can be achieved by invocation of the *Monitoring* interface operation *getMonitoringParameters()*. The *ParameterList* object returned as the result contains description of names of status parameters and their respective types.

To get the current state of the SDO, the observer has to invoke the operation *getCurrentStatus(),* which returns the list of parameter names and their values as *NVList* object. Knowing the respective types of status parameters, their values can be interpreted properly.

If the observer is interested in the current value of some particular status parameter, it can inquire information concerning the parameter of interest. This can be done by the invocation of *getParameter()* operation with the string name of the parameter as argument. The returned value should be interpreted according to the parameter type.

In general, any parameters that can change its value after certain period can be considered as monitoring parameters. The temperature value of the *Thermometer* SDO is a good example of a monitoring parameter. However, an SDO can decide which parameters can be monitored.

4.5.2.7 Configuration Interface

Each SDO or service can possess a unique set of configuration parameters, which can be read and modified through the *Configuration* interface. SDOs and *SDOService* classes that want to provide configuration operations for its parameters have to implement the *Configuration* interface.

Furthermore, *SDO* and *SDOService* classes that can store multiple configuration sets should implement the *ConfigurationExt* interface instead of the much simpler *Configuration* interface.

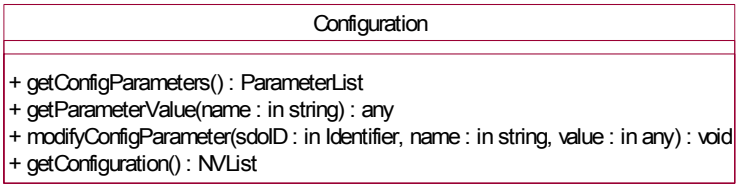


Figure 97: Configuration Interface

Operations provided by the Configuration Interface

Operation	getConfigParameters() : ParameterList
Return Value	The list with definitions of parameters characterizing the configuration
Description	This operation returns a list of all configuration parameters.

Operation	getParameterValue(name : string) : any
Return Value	The value of the specified parameter
Parameters	String - Name of the parameter whose value is requested
Description	This operation returns the current value of the parameter whose name is specified as argument. The parameter value is obtained as value of type any, so, depending on the implementation technology, it should be transformed to the appropriate type.

Operation	modifyConfigParameter(sdoID: Identifier, name : string, value : any)
Parameters	sdoID - Unique ID of the SDO modifying the parameter value. This value is used to check if the value can be modified when a request to modify service parameter is received Name - The name of the configuration parameter to be modified Value - New value of the specified parameter
Description	This operation sets the specific configuration parameter to the given value. Configuration parameter value may influence a service invocation. So when a service is reserved or invoked its configuration should not be changed by other SDOs. When a request to modify service configuration parameter is received, the interface validates if the modification request was sent by the SDO, which has invoked or reserved the service. If it is the value is modified, else exception is raised.

Operation	getConfiguration() : NVList
Return Value	The list of stored configurations with their current values
Description	This operation returns the current configuration that is the list of configuration parameters and their current values.

Usage of the Configuration Interface

The Configuration interface offers the possibility to configure the properties of an SDO. The implementers of each SDO can decide which object properties will be configurable through the Configuration interface. Since both the *SDO* class and the *SDOService* class implement this interface, an SDO and its services can be configured independently.

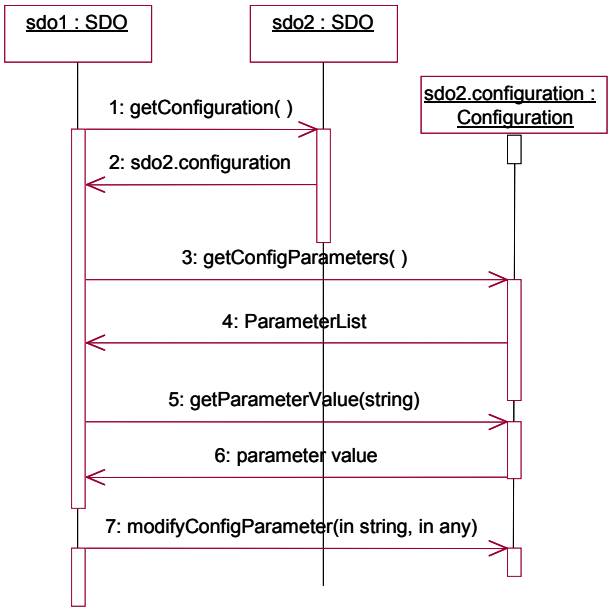


Figure 98: Message Sequence Chart: Configuration of SDO Properties.

The configuration of SDO properties is shown in Figure 98. First, the configuring SDO (*sdo1*) issues a request to get the `Configuration` object reference from the SDO that is about to be configured (*sdo2*). This request is shown as message 1. When the object responsible for the configuration is obtained (with message 2), *sdo1* inquiries the parameters characterizing the configuration of *sdo2* (message 3), and receives the list of names and types of properties in response (message 4). If *sdo1* is interested in current value of some particular property of *sdo2*, it requests its value (message 5) and can subsequently modify it (message 7).

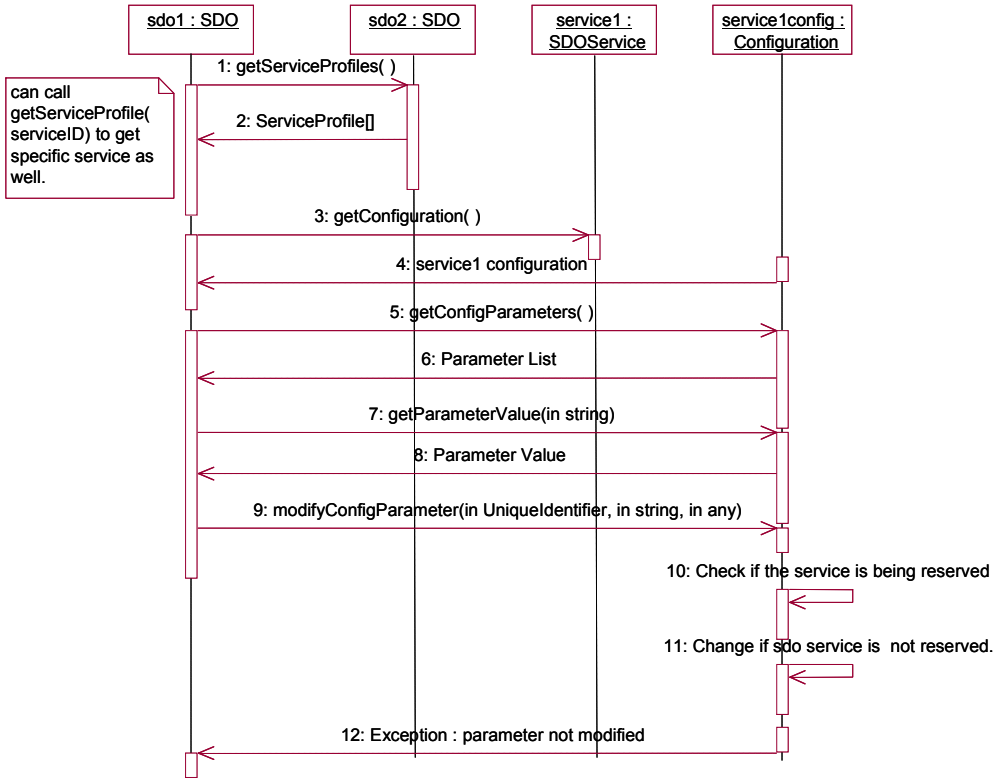


Figure 99: Message Sequence Chart: Configuration of Service Properties

The configuration of service properties is depicted in Figure 99. In this message sequence chart, the configuring SDO (*sdo1*) does not have any information about the services offered by the

participating SDO (*sdo2*). So it requests the list of service profiles first (message 1). This step does not actually belong to the configuration interaction and can be left out if the *sdo1* knows what service it is going to configure. Back to the sequence chart in Figure 99, after *sdo1* has chosen the appropriate service (*service1* of *sdo2*), it requests the list of parameters characterizing the configuration of this service (message 3).

If *sdo1* is interested in the current setting of some particular parameter, it inquires the service for the respective parameter value (message 5). The value of the parameter can be modified (message 7). Operations of obtaining current parameter values and of value modification can be invoked independently from each other. For example, the service of the *TemperatureController* SDO defines its parameters *minTemp* and *maxTemp* as configuration parameters. Hence, this service must provide an implementation of the *Configuration* interface.

4.5.2.8 ConfigurationExt Interface

Additionally to the *Configuration* interface, an SDO may implement the *ConfigurationExt* interface. This interface allows other SDOs to store and load several configuration parameters at once, denoted as stored configurations. A stored configuration contains the list of configuration parameters and their respective values. Each stored configuration can be activated individually, thereby all configuration parameters included in that stored configuration are restored.

For example, the *Airconditioner* SDO can provide this interface that other SDOs can store a configuration. In this case the *Thermometer* SDO can store different configuration settings, for example one to turn the heater *on* with *low* power, and another to turn the cooler *on* in *high* power. In case the temperature of the room goes below the defined threshold, the *TemperatureController* SDO can activate the configuration setting to turn the heater *on* in *low* power.

Data Structures defined for the ConfigurationExt Interface

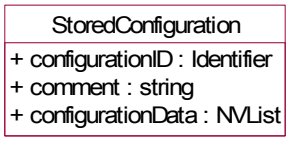
Name	Description
StoredConfiguration	 The structure of the stored configurations of an SDO or a service. *configID* – Identifier of the stored configuration. This identifier value is used to activate the stored configuration. The identifier is unique to this specific SDO only. *comment* – A brief, human readable description of the stored configuration. *configurationData* – List of stored configuration parameters with their respective values.
StoredConfigurationList	List of all stored configurations.

Table 14: ConfigurationExt Interface Data Structures

Operations provided by the ConfigurationExt Interface

Operation	getStoredConfigList() : StoredConfigurationList
Return Value	The list of stored configurations
Description	Gets all stored configurations of the object (an SDO or a service)

Operation	activateConfig(configID : Identifier)
Parameters	configID - The identifier of the configuration that should be activated
Description	The operation restores the specified stored configuration. Please note, that an exception is raised if the service is currently reserved or busy

Operation	addNewConfig(configuration : StoredConfiguration) : Identifier
Return Value	The identifier of the stored configuration
Parameters	Configuration - The configuration to be stored
Description	The operation adds new configuration settings.

Operation	modifyConfig(configID : Identifier, values : NVList)
Parameters	configID - The identifier of the configuration that should be modified values - List of configuration parameters with their new values
Description	This operation modifies the specified stored configuration. Thereby, the second argument contains a list of parameters that have to be modified along with their new values. This list can include the whole set of configuration parameters defined for this stored configuration or an arbitrary subset of these parameters.

Operation	removeConfig(configID : Identifier)
Parameters	configID - The identifier of the stored configuration that should be removed
Description	The operation removes the specified stored configuration from the stored configuration.

Usage of the ConfigurationExt Interface

The *ConfigurationExt* interface offers the capability to define and store configuration sets that can restore several configuration parameters at once. The configuration of an SDO with involvement of stored configurations is shown in Figure 100.

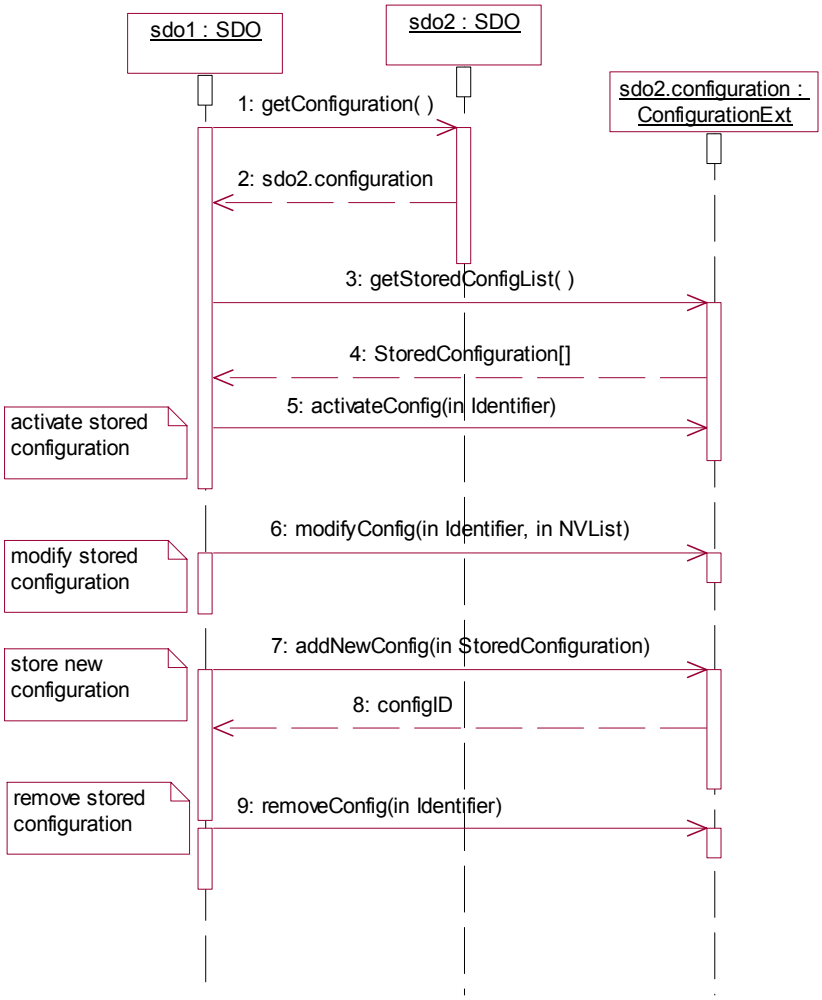


Figure 100: Message Sequence Chart Configuration: Stored Configurations Usage

As shown in Figure 102, the first step for an SDO (*sdo1*) configuring another SDO (*sdo2*) is to get *Configuration* object from the latter (message 1). Thereafter *sdo1* requests the list of configurations defined for *sdo2* (message 3). Then, it can pick out one of the stored configurations for retrieval (message 5) or modify the parameter values the configuration (message 6). If none of the stored configurations suffices its intents, *sdo1* can define a new configuration to be added

to the set of stored ones (message 7). Configurations that are no longer useful can be removed from the set (message 8).

SDO should keep track of all the stored configuration setting in a table with setting ID and its property values. The following table illustrates how two configurations are stored by the *Air-conditioner* SDO.

Configuration Paramter	Set 1	Set 2
configurationID	config_1	config_2
comment	Heater on	Cooler on
configurationData		
operationMode	heaterON	coolerON
roomConditioner_Service1.power	low	
roomConditioner_Service2.power		high

If a new configuration is stored, its value is added in the table. All the given SDO configuration parameters are stored with their values in the table. In example 1, value of the parameter *roomConditioner_Service2.power* and in example 2, value of the parameter *roomConditioner_Service1.power* is not required. It can store the values or just ignore it.

4.5.2.9 Reservation Interface

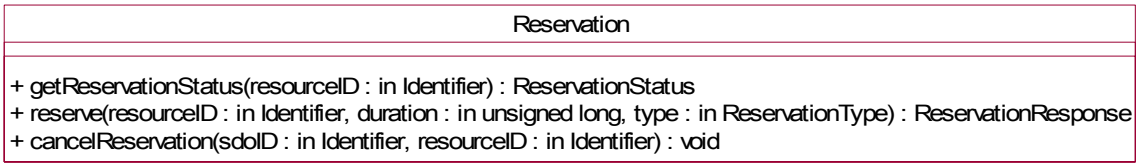
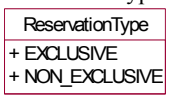
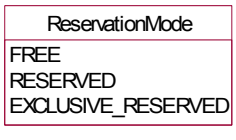


Figure 101: Reservation Interface

Data Structures defined for the Reservation Interface

Name	Description
ReservationType 	The enumeration defining the types of reservation supported. - *EXCLUSIVE* - The service is reserved exclusively to the reserving SDO, and no other SDOs in the network can either reserve or execute the service. The other SDOs cannot change its configuration parameters as well. - *NON_EXCLUSIVE* – The service is reserved non-exclusively to the reserving SDO. The reserved service can be used or reserved in non-exclusive mode by other SDOs in the network. However, an exclusive reservation is not accepted. Note: While being reserved in no-exclusive mode, a service can be accessed both by SDOs that have non-exclusive access rights and by SDOs that have made no reservation before. No SDOs including the SDOs that have reserved the service in this mode can configure its properties.
ReservationMode 	The enumeration defining the sate of SDO services regarding reservation. It is used to indicate the type of reservation other SDOs have on the SDO service. - *FREE* – The SDO service is not reserved. i.e. it can be used or reserved. - *RESERVED* – The SDO service is reserved with NON_EXCLUSIVE reservation type. - *EXCLUSIVE*_RESERVED – The SDO service is reserved with EXCLUSIVE reservation type.
ReservationStatus	 The structure specifies the status of current reservation. - *mode* – Mode of the reservation.

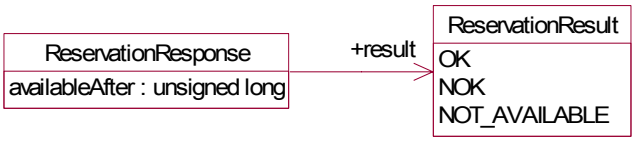
Name	Description
	- *availableAfter* - Time interval after which the reservation ends. Necessary only if the service is reserved.
ReservationResult	Enum ReservationResult {OK, NOK, NOT_AVAILABLE} Possible results for a request to reserve a service. It indicates if the reservation was made for the corresponding reservation request. - *OK* - The service has been reserved.. - *NOK* - The service could not be reserved. There are two possibilities for this return value: either the service is already reserved in EXCLUSIVE mode or it is reserved in NON_EXCLUSIVE mode but the caller requested EXCLUSIVE mode. - *NOT_AVAILABLE* - The service being reserved is currently not available.
ReservationResponse	 The structure *ReservationResponse* defines the data received in response to a reservation request. The field *result* indicates if the reservation succeeded. If the service is currently reserved, the response field *availableAfter* specifies the remaining time of reservation.

Table 15: Reservation Interface Data Structures

Operations provided by Reservation Interface

Operation	getReservationStatus(resourceID : Identifier) : ReservationStatus
Return Value	Reservation status of the specified resource or the service specifying the reservation mode of the service and the period after which the service will possibly be free if it is being reserved.
Parameters	resourceID - The identifier of the service whose reservation status is requested
Description	This operation gets the details of current reservation status of the specified resource or service.

Operation	reserve(resourceID : Identifier, duration : long, type : ReservationType) : ReservationResponse
Return Value	The response to reservation request received
Parameters	resourceID - The identifier of the service that should be reserved with this request duration - The time period until which the service reservation is requested Type - The type of reservation to be made
Description	This operation sends a request to reserve the specified SDO service. The argument resourceID specifies the identifier of the respective service. The argument duration defines the period for the reservation. The argument mode specifies the desired mode of reservation. The returned data structure of type ReservationResponse contains the response details. There are several possibilities of response to reservation request: If the result OK is passed back, the reservation has succeeded. The reservation can be identified with the identifier of the reserving SDO. The identifier can be used for invocation of service operations, or for cancellation of the reservation. The server returns NOK as result. The response points out that the reservation is not possible (for example, if the service is currently reserved exclusively) or cannot be accepted with proposed conditions (for example, if the service is reserved non-exclusively at the request time, and reservation in exclusive mode is being requested). In this case, the response field availableAfter specifies the time interval after which next reservation attempt can be undertaken (because the current reservation will end by then). The server returns the result NOT_AVAILABLE. This response indicates that the service whose reservation is requested is currently not available, for example, if it is blocked by operation execution, but can become available later.
Operation	cancelReservation(sdoID : Identifier, resourceID : Identifier)
Parameters	sdoID - Unique ID of an SDO canceling its reservation resourceID - ID of the resource (service) whose reservation should be cancelled
Description	This operation cancels an existing reservation of the specified SDO of the specified service.

Usage of the Reservation Interface

The message sequence charts below show process of service reservation and different event series that can follow the reservation request. In the sequence chart, depicted in Figure 102, it is assumed that the client SDO (*sdo1*) does not have any information about the services of the provider SDO (*sdo2*) at the beginning of their interaction. Therefore, in the first step it requests the list of service profiles of the provider SDO (message 1). Then it makes a request to get the `Reservation` object from the SDO (message 3). When the `Reservation` object is obtained, it attempts to reserve the service of interest by invocating `reserve()` with the appropriate service identifier, desired reservation duration, and reservation mode as arguments (message 5), and receives the response from server.

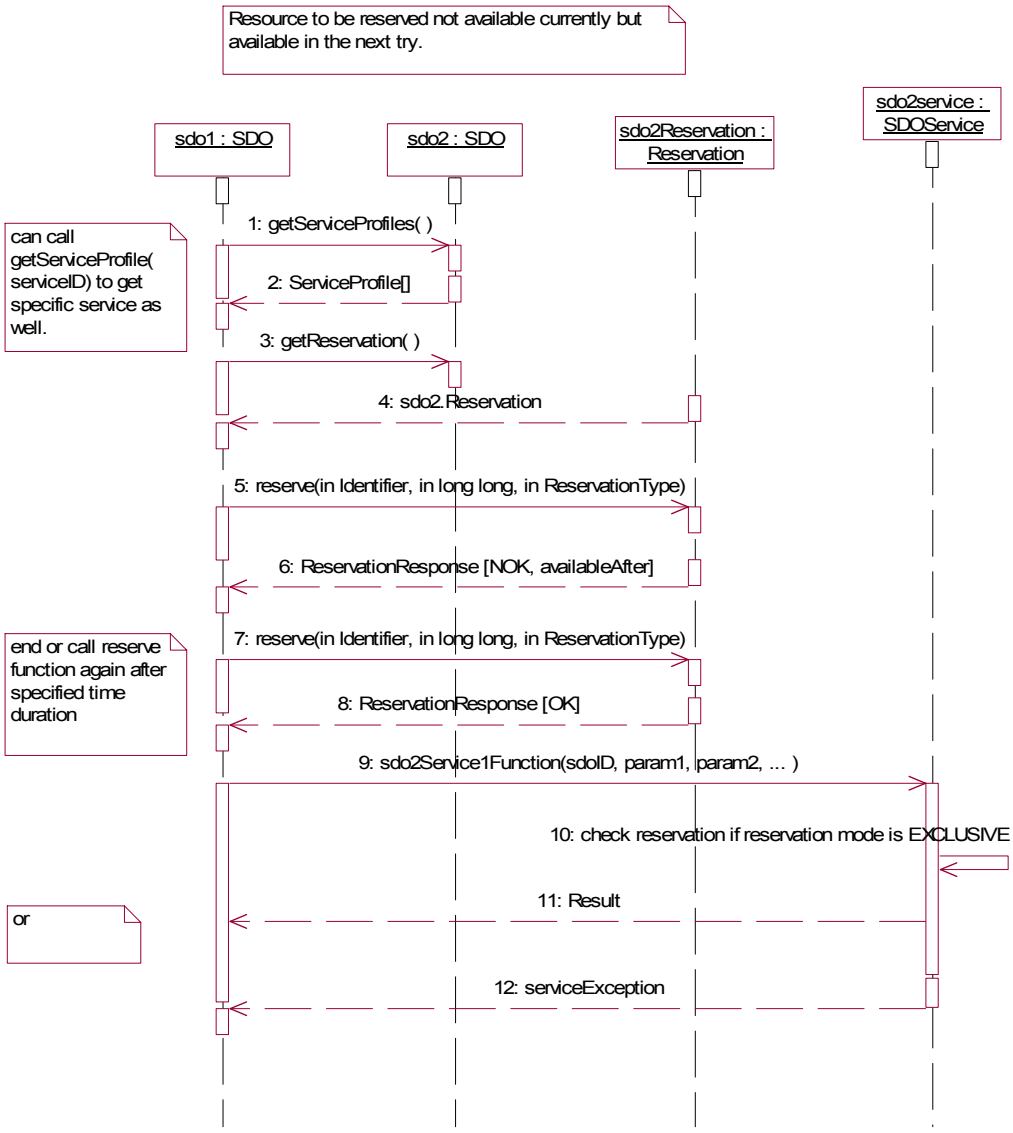


Figure 102: Message Sequence Chart Reservation: Reservation Request Denied

If the reservation succeeded, the server responds with OK (message 8). From now until the end of the reservation period, the client can use service operations exclusively, authenticating itself with its identifier, as it is exemplified with message 9. When calling the service operation, the client supplies its identifier as the first argument in order to evidence its right to use the service.

As response to service invocation, execution result or exception note can be returned (messages 10 and 11).

If the reservation is rejected, the server responses with NOK (message 6). The response message contains further information that can help the client to decide how to proceed with its own operations. Thereby, the field `availableAfter` specifies the period after which the current reservation expires. If the client will attempt to reserve the service after this period, the reservation will likely succeed (messages 7, 8).

If the client knows which service it is interested in at the very beginning of the reservation process, it can leave out step 1 and begin with getting the `Reservation` object directly. Moreover, if the client already possesses this object (for example, because it got the object earlier), step 3 can be omitted as well. The sequence chart depicted in Figure 103 shows an example where the reservation request is rejected because the requested service is not available at that time. In this case, the server answers with the response `NOT_AVAILABLE` (message 6). It is left to the client, if it will attempt to reserve the service later.

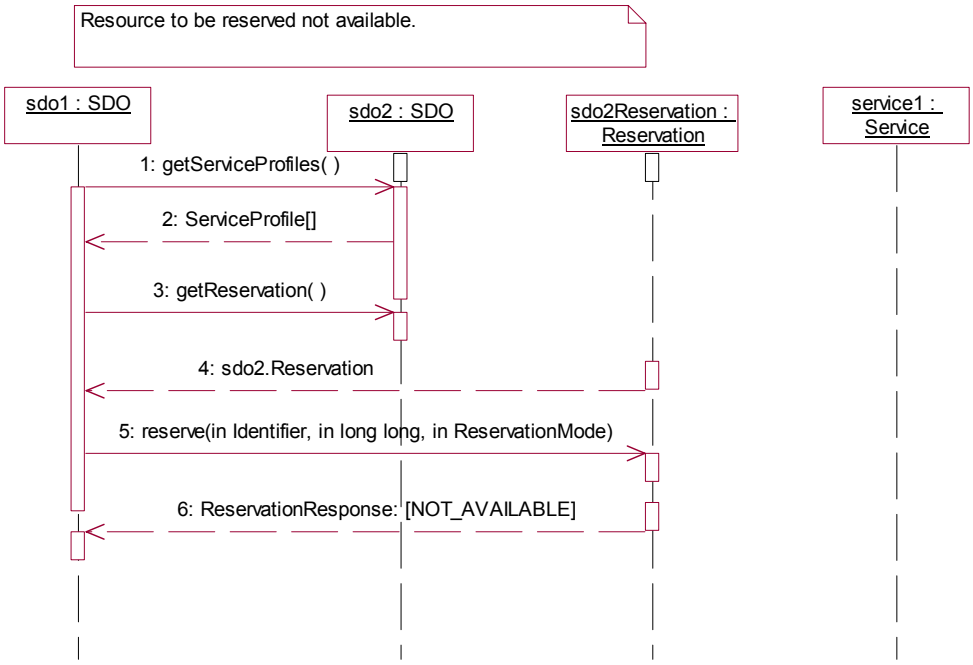


Figure 103: Message Sequence Chart Reservation: Resource not Available

4.5.3 General Implementation Aspects of SDOs

This chapter describes the implementation of the SDO specification, which was done in order to provide a proof of concept. It contains a detailed description of the implementation, describes the selected target platforms, and discusses possible mappings of the SDO specification to them.

Furthermore, it is described detailed how an SDO can be created using the prototype implementation. This chapter starts describing the implementation of the SDO specification using the CORBA platform.

4.5.3.1 SDO implementation on CORBA Platform

This section describes the current SDO implementation, which is based on the CORBA platform. Initially it explains what has been implemented. Then the overview of the implementation is given followed by the detailed description. It also provides a guideline to create new SDOs and gives an overview on the demonstration scenario.

The *SDO*, *Discovery*, *Monitoring*, *Configuration*, and *ConfigurationExt* interfaces as defined in the CORBA PSM have been implemented as well as a number of necessary helper classes including a basic graphical user interface (GUI). Several exemplary implemented SDOs and two demonstration scenarios showing the feasibility of the implementation complete the realization of the CORBA PSM.

The current implementation provides a modular approach to the realization of the CORBA PSM. The *Monitoring*, *Configuration*, and *ConfigurationExt* interfaces are realized as components whose implementations can be changed independent of the SDO implementation if necessary. The Reservation interface from the CORBA PSM has not been implemented.

4.5.3.2 SDO Package

The SDO implementation is based purely on Java. It uses the CORBA implementation (ORB) supplied with the JDK 1.4.1. The implementation consists of five parts:

- the core *SDO* interface including the *Discovery* interface,
- the *Monitoring* and *Configuration(Ext)* interfaces,
- the *NotificationDaemon*,
- the SDO configuration, and
- the basic SDO GUI.

The implementation is organized in several packages:

- The package *de.fhg.fokus.sdo.shared* contains the *AbstractSDO*, *Monitoring* interface, *Configuration(Ext)* interface, and *NotificationDaemon* implementations, and several internal helper classes beginning with *Internal**.
- *de.fhg.fokus.sdo.shared.preferences* contains all classes for the SDO configuration.
- All classes necessary for the basic SDO GUI belong to the package called *de.fhg.fokus.sdo.shared.gui*.
- The package *de.fhg.fokus.sdo.shared.corba* contains a CORBA utility class utilized by several other classes.
- *de.fhg.fokus.sdo.shared.sdobase* contains the automatically created CORBA stubs and skeletons.

The implementation is centered on the abstract core class *AbstractSDO* that implements the *SDO* and *Discovery* interfaces and provides an internal interface to other components for accessing internal SDO attributes and functions. As depicted in Figure 104, a concrete SDO implementation, e.g. a light SDO, has to extend the *AbstractSDO class*. *AbstractSDO* provides two methods to get the *Monitoring* and *Configuration* interfaces of an actual SDO – *getMonitoring()* and *getConfiguration()*.

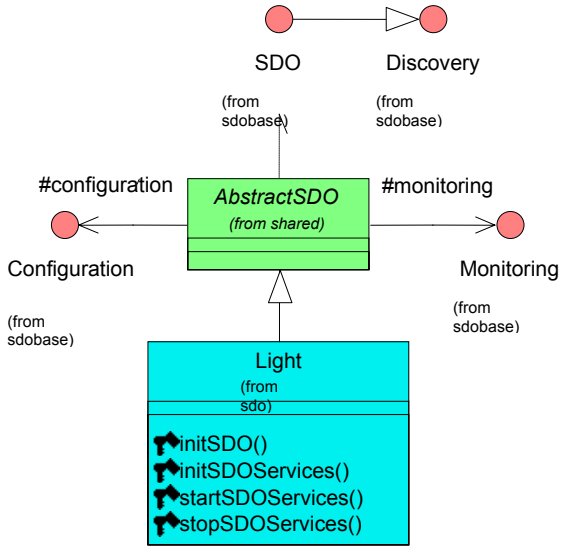


Figure 104: Core Class AbstractSDO

The *Monitoring* and *Configuration* interfaces are internally represented by the two abstract classes *Configuration* and *Monitoring*. Default implementations for these interfaces are provided with this implementation – the *AbstractMonitoring* and *AbstractConfiguration* classes. The actual *Monitoring* and *Configuration* interface implementations are instantiated during runtime at the SDO initialization. These objects are initialized with an instance of the internal interface of the core SDO object.

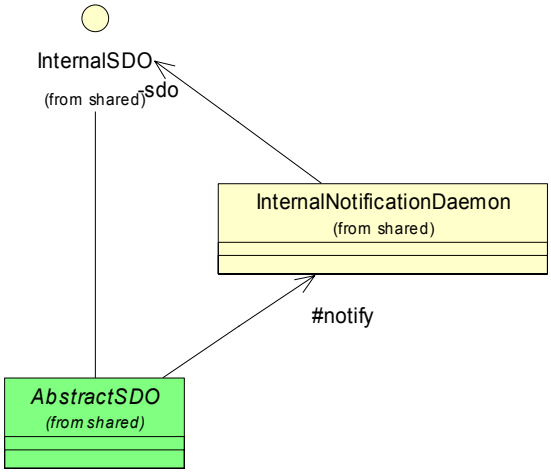


Figure 105: Notification Demon

The notification and subscription part of the *Monitoring* interface relies on an internal notification daemon, which manages all subscriptions for specific SDO parameters that should be actively monitored. The daemon also manages the current state of the subscription renewals and notifies all subscribers when necessary.

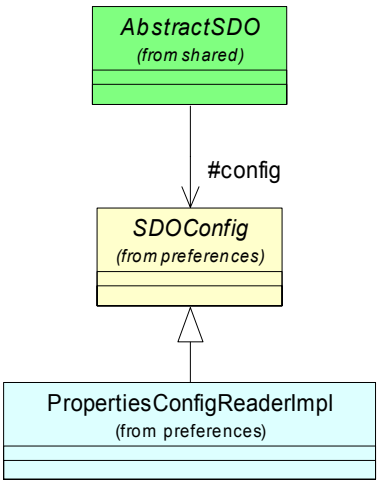


Figure 106: Internal Abstract Configuration Class

The initial SDO configuration, which includes SDO parameters, service profiles, device profile, and names of implementations for the core SDO class, the abstract *Monitoring* interface, and the abstract *Configuration* interface, is provided by the internal abstract configuration class *SDOConfig*. The supplied implementation of this class *PropertiesConfigReaderImpl* reads an external property file.

The extension of the *Configuration* interface the *ConfigurationExt* interface has been implemented as well. It is internally represented as an abstract class (*AbstractConfiguration-Ext*). A default implementation for this interface is provided with this implementation.

4.5.4 Detailed Implementation Description

This section describes the implementation of the CORBA Platform Specific Model (PSM) in more detail. In the beginning, the realization of the *SDO* and *Discovery* interfaces is shown. Following that, this section describes the internal class responsible for the provisioning of the initial SDO setup parameters. Subsequently the realization of the *Monitoring* and *Configuration* interfaces is depicted. This section concludes with the description of the basic SDO GUI provided for all further SDO implementations. The complete specification of the PSM is given in OMG IDL in section 7.5.

4.5.4.1 SDO Interface Implementation

The *SDO* and *Discovery* interfaces are implemented as abstract classes, which additionally provide

- an interface for accessing internal SDO attributes and functions, that is used by other internal components like the *Monitoring* interface, and
- four abstract methods, that must be implemented by actual SDOs.

This abstract class is called *AbstractSDO*. The internal interface, which is implemented by the class *AbstractSDO*, is called *InternalSDO*. The class *AbstractSDO* builds the core of the CORBA PSM implementation. All other components are centered around this class. Individual SDOs are realized by extending this class and implementing at least the four abstract methods provided by this class.

The next two sections describe the class *AbstractSDO* and its interface *InternalSDO* in more detail.

AbstractSDO

The abstract class *AbstractSDO* implements the *SDO* and *Discovery* interfaces described in the SDO IDL. It uses an SDO*Config* object to load the initial SDO setup during the SDO initialization. The SDO initialization is done in the static method *startup()*, which is called from the class' main method. It initializes some helper classes, loads the initial SDO setup, instantiates the actual SDO implementation class (e.g. *Light*), calls the methods *initSDO()* and *initServices()* in the SDO implementation class, and finally starts all services offered by the actual SDO. This is done via the method *startSDOServices()* in the SDO implementation class.

The message sequence chart in the next figure shows the initial SDO setup more detailed.

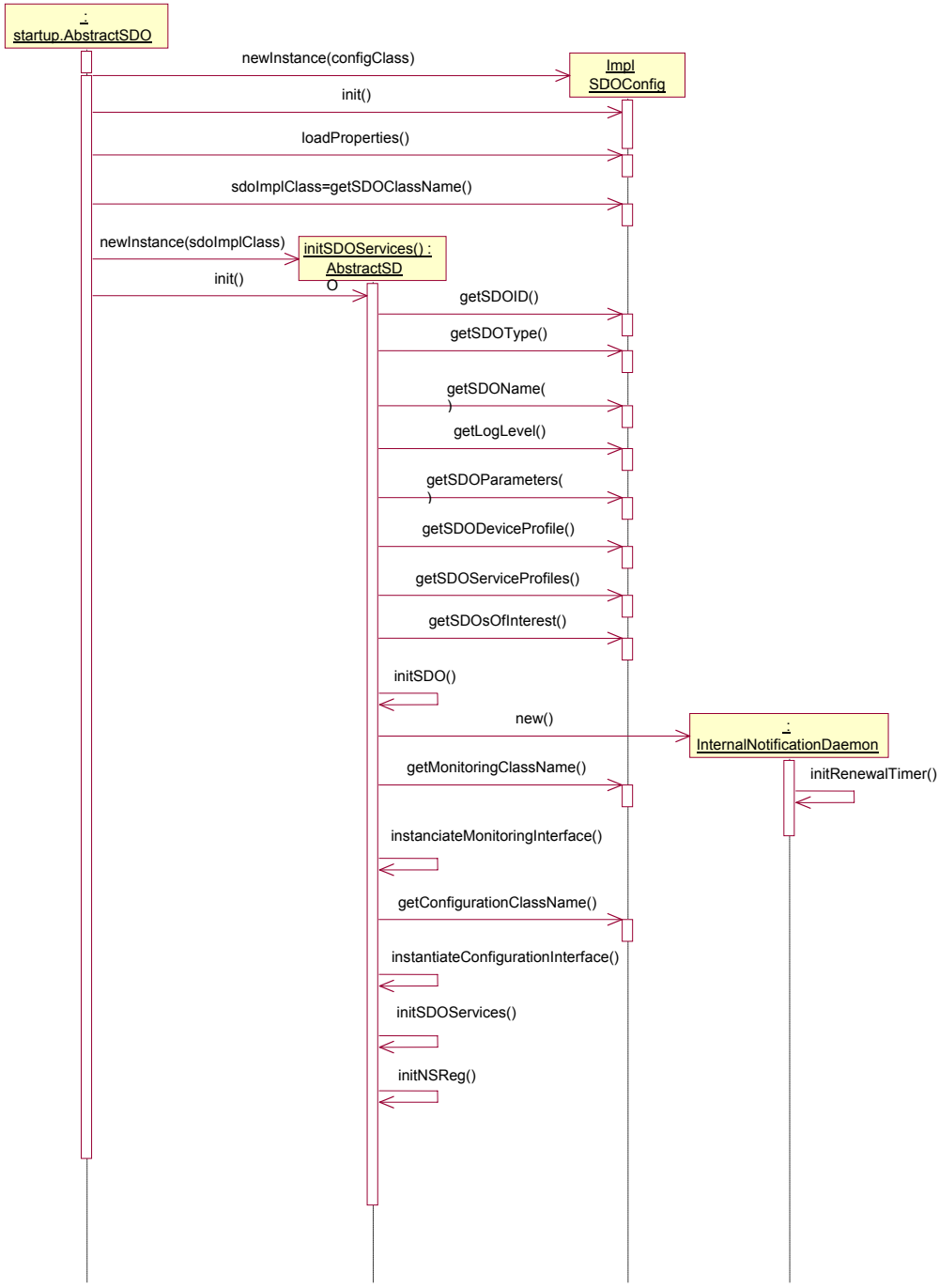


Figure 107: MSC: Initial SDO Startup

The *AbstractSDO* contains internal fields including the actual SDO name, id, type, and parameters as well as instances of the *Monitoring*, *Configuration* or *ConfigurationExt* interfaces. Figure 108 shows the internal fields used in the class *AbstractSDO*.

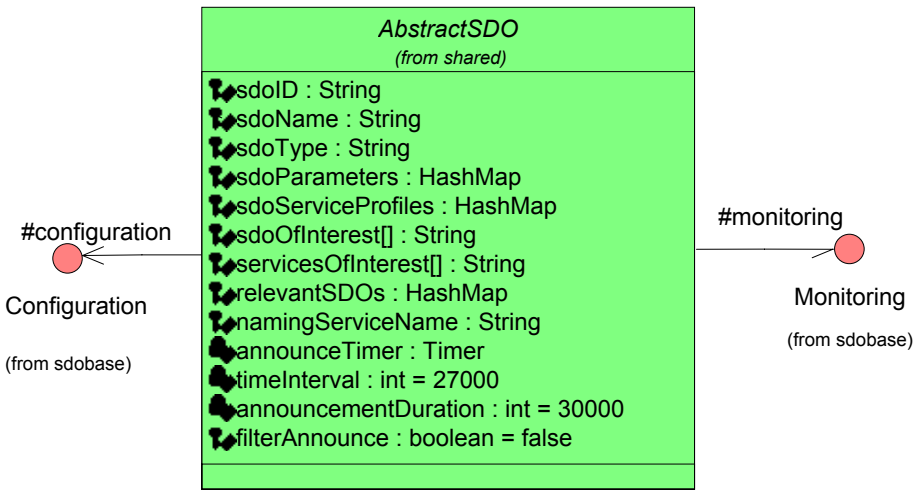


Figure 108: Class AbstractSDO

Actual SDO implementations need to implement at least the four methods shown in Figure 109. The method *initSDO()* allows actual SDO implementations to execute some additional steps during initialization, e.g. setting the current value of an SDO location parameter. The SDO implementation class can perform the initialization of its offered services in the method *initSDOServices()*. This method is called during the SDO startup controlled by the class *AbstractSDO*.

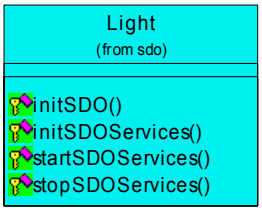


Figure 109: Methods of the Class AbstractSDO to be implemented by actual SDOs

At the end of the SDO startup the method *startSDOServices()* is called to start the services offered by the actual SDO.

When the method shutdown from the *InternalSDO* interface is called in order to gracefully stop the SDO's execution, the method *stopSDOServices()* is called to stop all previously running services.

The *Discovery* interface is partially implemented in the class *AbstractSDO*, namely the operation *announce()* is used by SDOs to publish messages about their presence. The principles of discovery are defined in section 3.3.3.3. The implemented announcement mechanism works as described below.

After the SDO interfaces and services are initialized, the operation *prepareAnnounceMsg()* is called. The SDO composes an announcement message and calls the operation *sendAnnouncement()*. This operation reads the entries in the CORBA Naming Service directory that are placed under the naming context '.SDO' and gets the object references bound to them. The objects registered under this context are SDOs. On each of them the SDO invokes the operation *announce()*. As the operation argument, the SDO announcement is used. The processing of incoming announcements by SDOs is described later. After the first `NEW` announcement is sent, the SDO starts a timer. The timer ensures that the `ALIVE` announcements are sent henceforth regularly as defined in the *AbstractSDO*.

When the SDO is shut down, an announcement containing a `LEAVE` message is composed and sent to all SDOs that are currently registered with the CORBA Naming Service.

Processing of incoming announcements: SDOs, that execute *announce()* operation, check if the announcement comes from an object of interest. If it is not so, the announcement is ignored; otherwise the operation *processAnnouncement()* is executed. In this operation, the SDO evaluates the contents of the announcement. If the announcement contains a `NEW` message, the information about the sender and its object reference is saved in the list of relevant SDOs. A list entry contains also a field with the lifetime of the announcement. In case of an `ALIVE` announcement, the lifetime of the announcement is prolonged. If the announcement contains a `LEAVE` message, the data on the corresponding SDO is removed from the list of relevant SDOs.

Figure 110: SDO Discovery Mechanism

The search operations defined in the Discovery interface, namely *searchSDO()* and *searchSDOService()*, are currently not implemented.

InternalSDO

The class *AbstractSDO* implements the *InternalSDO* interface, which provides access to internal SDO attributes and functions. All other internal SDO components like the *Monitoring*, *Configuration*, and *ConfigurationExt* interface implementations use the *InternalSDO* interface to access the abstract SDO implementation. This enables a better separation between the core SDO Interface implementation and additional components and interface implementations.

The *shutdown()* method is called by the SDO GUI and gracefully stops all services offered by the SDO and deactivates the CORBA servant. The class *InternalNotificationDaemon* provides functions for *Monitoring* and *Configuration* interface objects in order to allow them to manage subscriptions on internal SDO parameters. The daemon periodically checks whether specific subscriptions are still valid or not.

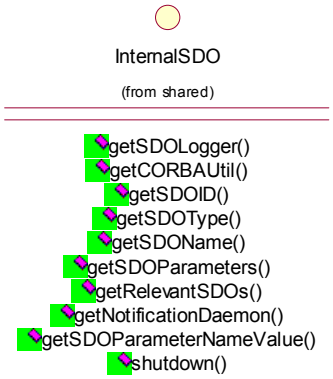


Figure 111: InternalSDO Interface

4.5.4.2 SDO Preferences

The static properties of an SDO can be configured through the implementation of the abstract class *SDOConfig*. It holds all configurable SDO properties in simple Java variables. This allows the development of new SDO properties readers without knowing the CORBA interfaces and internal SDO structures.

The name of the class implementing the properties reader has to be set in the Java system property *de.fhg.fokus.sdo.shared.preferences.SDOConfig*.

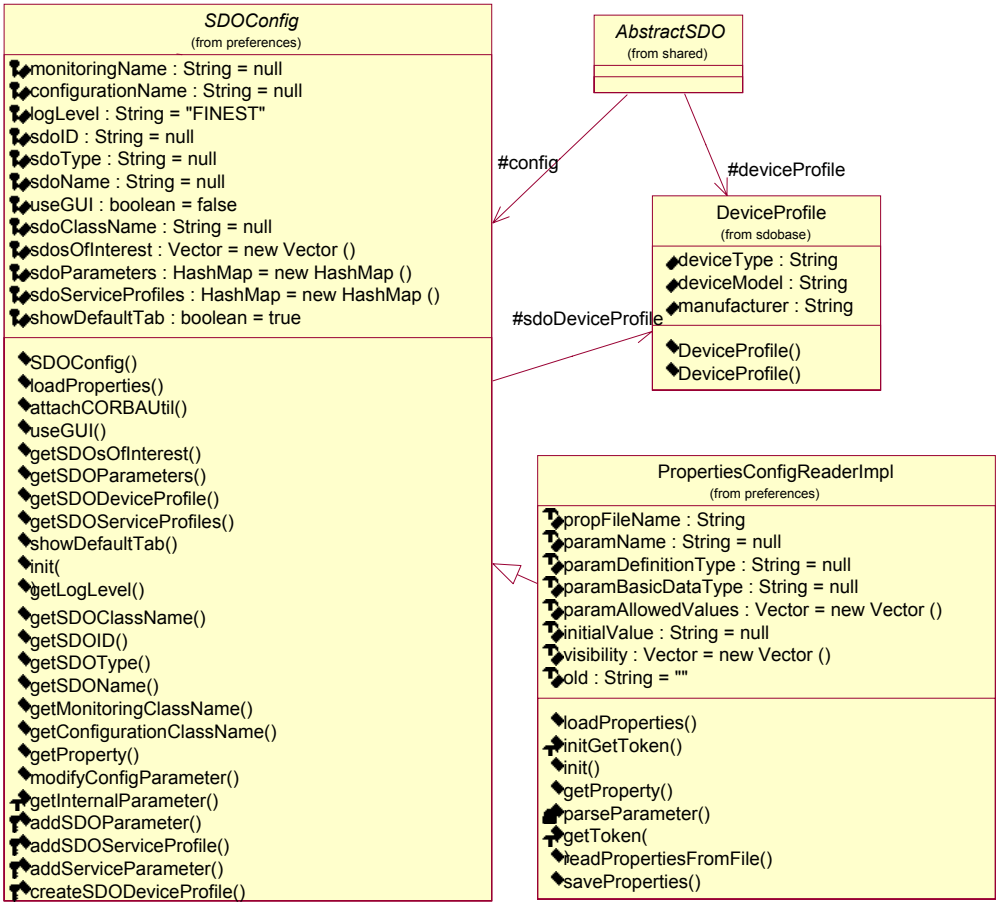


Figure 112: SDOConfig Class

The class implementing an SDO properties reader has to fill in some of these attributes directly and others through the call of one of these four protected methods:

- addSDOParameter(String name, String definitionType, String basicDataType, Vector allowedValues, String initialValue, Vector visibility)

This adds a parameter to the SDO. The following variables are passed to the method:

Variable	Description
name : String	The name of the SDO parameter.
definitionType : String	Type of the SDO parameter: short, long, float, double, string, enumeration, range or interval
basicDataType : String	The type of the values in a range or interval type.
allowedValues : Vector	The allowed values for an enumeration, range or interval type.
initialValue : String	The SDO parameter is set to this value initially.
Visibility : Vector	Can be monitorable, configurable or both.

- addSDOServiceProfile(String serviceID, String interfaceReference, String serviceReference, String serviceDescription)

This adds a service profile to the SDO. The following variables are passed to the method:

Variable	Description
serviceID : String	The unique ID of the service.
interfaceReference : String	Interface reference, e.g. for SDOs based on CORBA: *IDL:LightDevices/Switch:1.0*.
serviceReference : String	A reference to the service object this service profile is representing.
serviceDescription : String	A semantic description of the service, e.g. switch.

- addServiceParameter(String serviceID, String name, String definitionType, String basicDataType, Vector allowedValues, String initialValue, Vector visibility)
- Add a parameter to a service. The parameters of this method correspond to those of *addSDOParameter(),* except that you have to pass the service ID.
- createSDODeviceProfile(String deviceType, String deviceModel, String manufacturer, HashMap deviceSpecifications)
- Create an SDO device profile.

Variable	Description
deviceType : String	The type of device, e.g. light or radio.
deviceModel : String	The device model number given by the manufacturer.
manufacturer : String	The name of the manufacturer of the device.
deviceSpecifications : HashMap	*HashMap* defining the specifications or the characteristics of the device, e.g. name: voltage, value: 220.

With the help of these methods, the *SDOConfig* object provides SDO parameters, service profiles and a device profile to the *AbstractSDO*. By calling get methods for these attributes, the *AbstractSDO* fills its own data structures. This is done in the *init()* method of *AbstractSDO*.

The following attributes have to be set by the implementing SDO properties reader:

Attribute	Description
monitoringName : String	The name of the class implementing the monitoring interface.
configurationName : String	The name of the class implementing the configuration interface.
logLevel : String	The initial level for the logger.
sdoID : String	The unique identifier of the SDO.
sdoType : String	Semantic type of the SDO.
sdoName : String	Name describing the SDO.
useGUI : boolean	Set to true if the SDO uses a GUI.
showDefaultTab : boolean	Set to true if the SDO should show the default tab for managing the SDO, i.e. the standard graphical user interface.
sdoClassName : String	The name of the class implementing the SDO.
sdodOfInterest : Vector	A list of SDOs that are relevant for this SDO.

These attributes are also set in the *AbstractSDO* in *init()* through the usage of get methods. The whole process is illustrated in the figure above.

SDO Parameter

The SDO parameters are internally stored in objects of the type *InternalParameter*. This class contains the CORBA parameter structure, the value of the parameter and flags describing if the parameter is monitorable, configurable or both. The CORBA *Parameter* structure holds the name of the parameter, its CORBA type code, and the allowed values for *range* and *interval* types. The actual value of the parameter is stored in a CORBA *Any* variable.

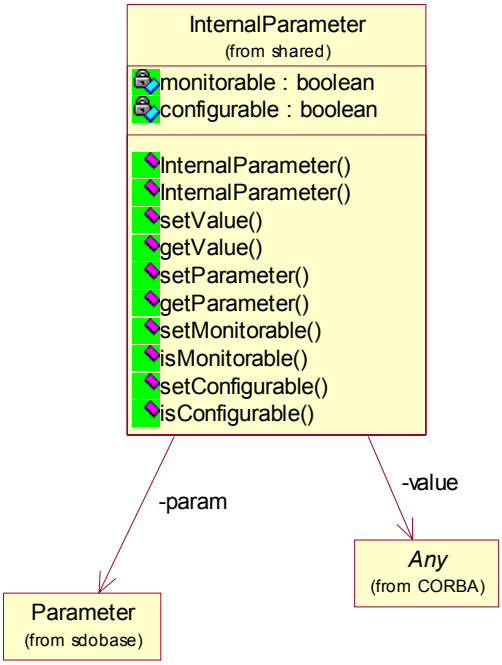


Figure 113: Class InternalParameter

SDO Service Profiles

The SDO *ServiceProfiles* describe SDO services. Like the SDO *Parameter*, there is an *InternalServiceProfile*, which contains the CORBA *SDOServiceProfile* structure. Additionally, it contains a *HashMap* of all parameters for the service.

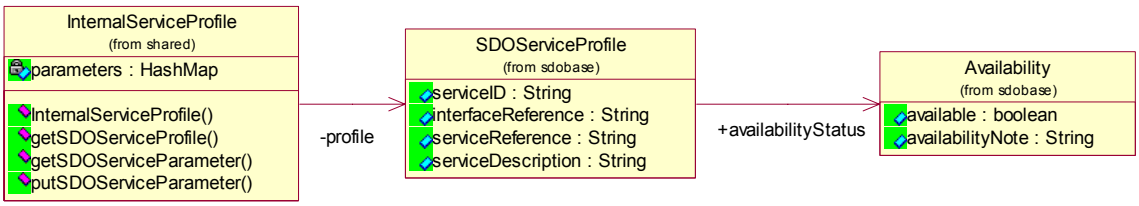


Figure 114. Class InternalServiceProfile

SDO Device Profile

The SDO *DeviceProfile* describes the device the SDO is representing. An SDO can describe none, e.g. if it is a service. Alternatively, it describes only one underlying device.

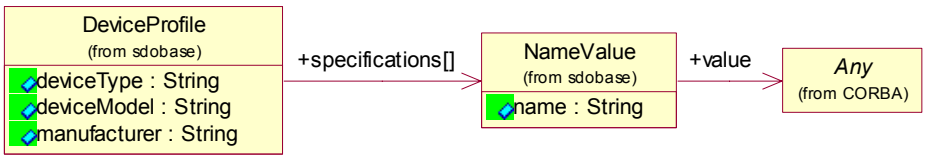


Figure 115: Class DeviceProfile

Properties Config Reader Implementation

The class *PropertiesConfigReaderImpl* is a default implementation of the *SDOConfig* class suited for Java properties files. The Java system property *de.fhg.fokus.sdo.shared. preferences.SDOConfig* is set to *de.fhg.fokus.sdo.shared.preferences.PropertiesConfig-ReaderImpl,* if it is empty when the system starts. An example of a property file is shown below.

```
# Implementation classes for SDO, Monitoring, and Configuration interfaces

SDO=de.fhg.fokus.sdo.Light
Monitoring=de.fhg.fokus.sdo.shared.MonitoringImpl
Configuration=de.fhg.fokus.sdo.shared.ConfigurationImpl

########### Properties of Light SDO at Home ####################

SDOID=Light_0adc
SDOName=Light on the ceiling
SDOType=LightSwitch

# DeviceProfile
DeviceManufacturer=Phillips
DeviceType=light
DeviceModel=LS-12
DeviceSpecifications=voltage,220;color,orange

# start with gui
# values: 'true' or 'false'
# default is 'false'
UseGUI=true
# show the default tab to access all relevant SDOs?
ShowDefaultTab=false

# set the debug level
DebugLevel=FINEST

# Log file name including the directory
# ie. LogFile.log
# if specified as above the log file will be stored as
SDO2_HOME\log\LogFile.log
# default is the '[sdoID].log' of the SDO
LogFile=Light_0adc.log


########### PARAMETER DESCRIPTION ####################

Param[1] = Location;string;;;Home;configurable
Param[2] = NotificationFrequency;short;;;10;configurable
Param[3] = LightState;enumeration;;on,off;on;monitorable
Param[4] = Color;enumeration;;blue,red;blue;monitorable


############ SERVICE DESCRIPTION ############################

ServiceSpec[1]          =lightservice;IDL:Devices/LightSwitch:1.0;;switch
ServiceParam[1][1]      =P1;enumeration;;on,off;on;monitorable
ServiceParam[1][2]      =P2;enumeration;;on,off;on;monitorable

ServiceSpec[2]          =light2service;IDL:Devices/LightSwitch2:1.0;;switch
ServiceParam[2][1]      =P3;enumeration;;on,off;on;monitorable
ServiceParam[2][2]      =P4;enumeration;;on,off;on;monitorable


# SDOs and services relevant to this SDO

RelevantSDOs     = Light
RelevantServices = switch
```

The names of the properties correspond largely to the attributes of *SDOConfig*. The SDO parameters are defined with *param* followed by the number of the parameter in square brackets. Equivalently, services and their parameters are specified.

4.5.4.3 Monitoring Interface

The implementation of the *Monitoring* interface is based on the abstract class *AbstractMonitoring* that implements the *SDOInterface* interface as well. This interface makes an *init()* method available in order to initialize and start the actual *AbstractMonitoring* implementation at the SDO startup. The class *AbstractMonitoring* extends the *MonitoringPOA* object. All *Monitoring* interface operations are declared abstract so that the actual implementation class has to provide them.

The method *initMonitoring()* from *AbstractMonitoring* could be overwritten in order to provide a custom initialization which might be necessary for an actual *Monitoring* interface implementation. The figure shown below depicts the relations between all necessary classes for the current implementation of the *Monitoring* interface.

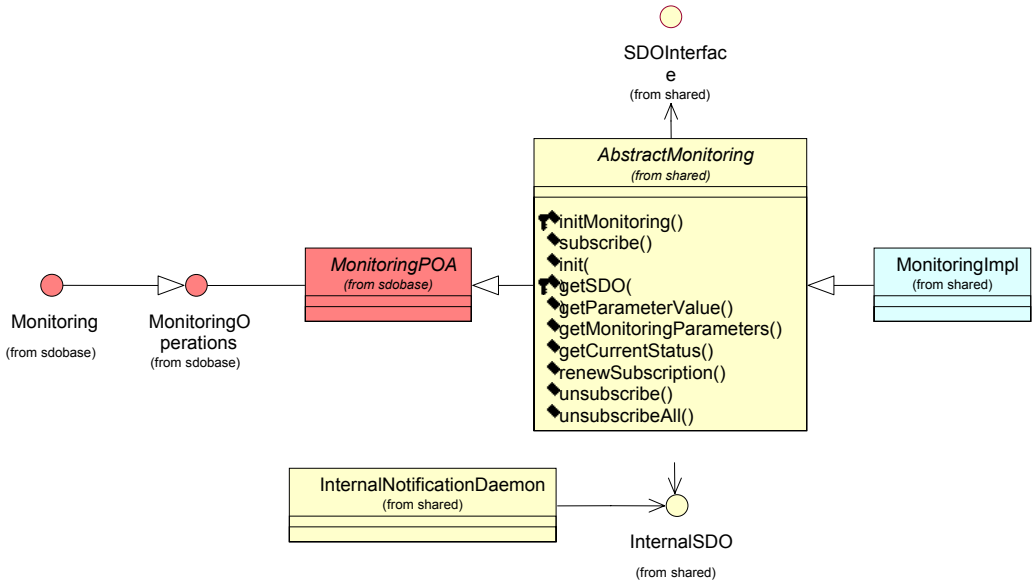


Figure 116: Monitoring Interface

The class *MonitoringImpl* provides an implementation of the *Monitoring* interface. All internal SDO parameters are accessed via an instance of the *InternalSDO* interface and the method *getSDOParameters()*. The *InternalSDO* object has been assigned at the initialization process by the *AbstractSDO*. This method returns a *HashMap*, which contains all SDO parameters as *InternalSDOParameter* objects and the SDO parameter names as keys. The *InternalNotifcationDaemon* is used in order to subscribe or to unsubscribe certain SDO parameters and to renew existing subscriptions.

4.5.4.4 Notification Daemon

The class *InternalNotificationDaemon* manages all *subscribe()*, *unsubscribe()*, and *renewSubscription()* requests for a particular SDO and its *Monitoring* interface. In addition to that, it implements an internal *renewalTimer*, which monitors periodically the renewal status of all registered subscriptions. When a particular subscription has expired, the *renewalTimer* calls the *removeSubscription()* method of the *InternalNotificationDaemon*. This method removes all entries, which have been previously associated with this particular subscription, in the *notificationParametersOnChange* and *notificationParametersOnInterval* attributes.

The *notificationParametersOnChange* map stores so-called notifications *HashMap* for each monitored SDO parameter. The parameter is identified by its unique name. The notifications *HashMap* holds for each subscriber a *Vector* with all registered subscriptions, i.e. it is possible for one subscriber to register for the same SDO parameter more than one time with different subscriptions respectively notification callbacks. The subscriber is uniquely identified by its id. This structure is built when a new subscription is registered with the method *addSubscription()*.

Like the *notificationParametersOnChange* map the *notificationParametersOnInterval* map stores a notifications *HashMap* for each periodically monitored SDO parameter. Utilizing both maps the method *parameterChanged(),* which is called from within the method *modifyS-DOParameter()* of the *AbstractSDO*, notifies all subscribers. The figure below shows the *Internal-NotificationDaemon* class diagram.

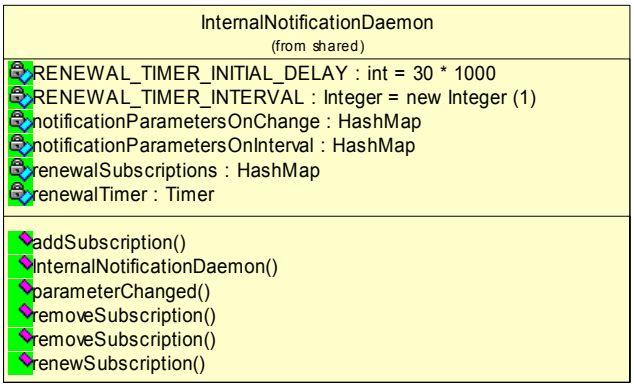


Figure 117: Class InternalNotificationDaemon

4.5.4.5 Configuration and ConfigurationExt Interfaces

The implementation of the *Configuration* and *ConfigurationExt* interfaces is based on two abstract classes, *AbstractConfiguration* and *AbstractConfigurationExt,* which implement the *SDO-Interface* interface as well. This is implemented similar to the the *Monitoring* interface.

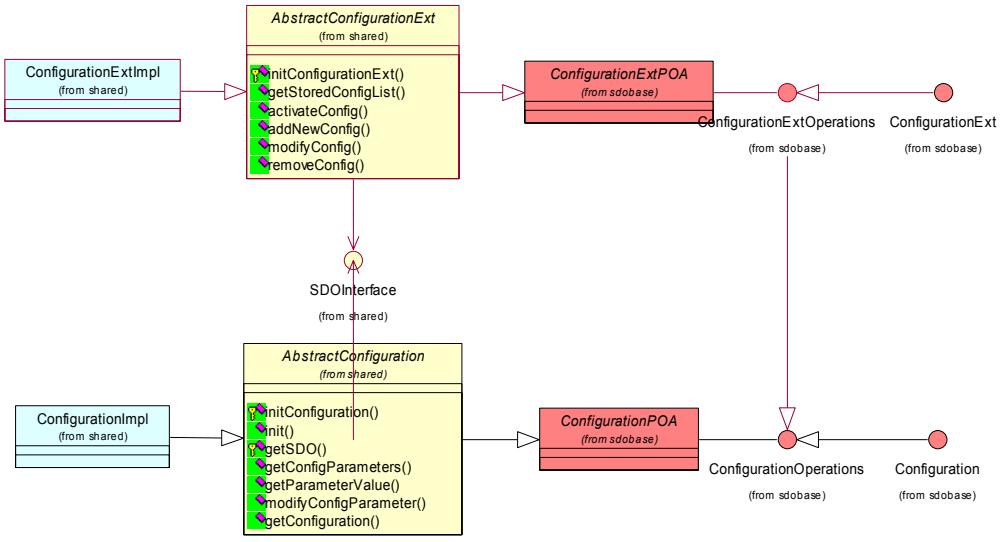


Figure 118: Configuration and ConfigurationExt Interfaces

The class *AbstractConfiguration* extends the *ConfigurationPOA* and the class *AbstractConfigurationExt* extends the *ConfigurationExtPOA*. All *Configuration* and *ConfigurationExt* operations are declared abstract so that the actual implementation classes have to provide them. The

ConfigurationImpl and *ConfigurationExtImpl* classes implement the abstract methods inherited from their parents. The *InternalSDO* object, which is provided at the interface initialization, is used to access all internal SDO parameters.

When a parameter is changed via the *modifyConfigParameter()* method, the *AbstractSDO* notifies all subscribers via its *InternalNotificationDaemon* instances.

4.5.4.6 SDO GUI

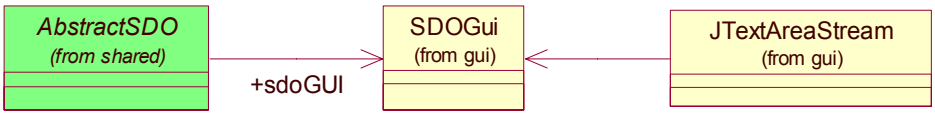


Figure 119: SDO GUI Implementation

The class `de.fhg.fokus.sdo.shared.gui.SDOGui` implements the basic graphical user interface for all SDOs. Thereby, the *SDOGui* class presents the necessary framework a specific SDO can use to display its own output. By default only generic information about the SDO as well as the logging output are displayed within the GUI. However, an SDO can and should provide additional panels to display its own output. See Figure 121 for a simple SDO that supplies an additional panel displaying a picture. The figure also displays the log output panel as provided by the generic *SDOGui*. However, the default panels can be disabled by setting `showDefaultTab` in `SDOConfig` to false through the configuration implementation.

Figure 120: Each SDO can add customized panels to the GUI.

The GUI is instantiated in `startGUI()` in `AbstractSDO`. Thereby, `startGUI()` is called in `startup()`. The GUI is only instantiated if `useGUI()` of `SDOConfig` returns true, e.g. if the property `UseGUI` is set to `true` in the property file of the SDO.

```
sdoPanel = new PictureViewerPanel(this);
getSDOGUI().addPanel(sdoPanel, "Picture");
```

If an SDO wants to add its own GUI to the standard GUI, it should add own panels to the main tab. This is done by creating a panel and calling the method *addPanel()* of *SDOGui* with the panel and a name for the tab as parameters.

Figure 121: Simple SDO with one additional Panel and Log output of the Simple SDO

4.5.5 SDO Maintenance Tools

Since there is no central server keeping a list of all SDOs in the network, each SDO merely knows its own state and the state of the SDOs its monitors. However, to get an overview of all SDO in the network a special ControlPoint SDO is included in the implementation of the SDO specification.

Figure 122: SDO GUI Screenshot

Through this SDO, as shown in Figure 122, an administrator can get information about all known SDOs in the network. All SDOs are listed in the tree view on the left side. The user can select an SDO to get general information or access special interfaces. If an SDO provides a *Configuration* interface, the `Configuration` node displays the name, the type, and the value of all configurable parameters. Changing a value is possible by double-clicking the configuration value. Similarly, if an SDO provides the *Monitoring* interface, all monitoring parameters are listed after selecting the `Monitoring` node. The user can subscribe to monitoring parameters to watch how their value changes. The `Service` node displays the services an SDO provides and allows to access their configuration parameters. The screenshot in Figure 123 shows the GUI of the *ControlPoint* SDO with the configuration panel selected for the *Light* SDO. Furthermore, Figure 123 displays the monitoring panel of the light switch SDO.

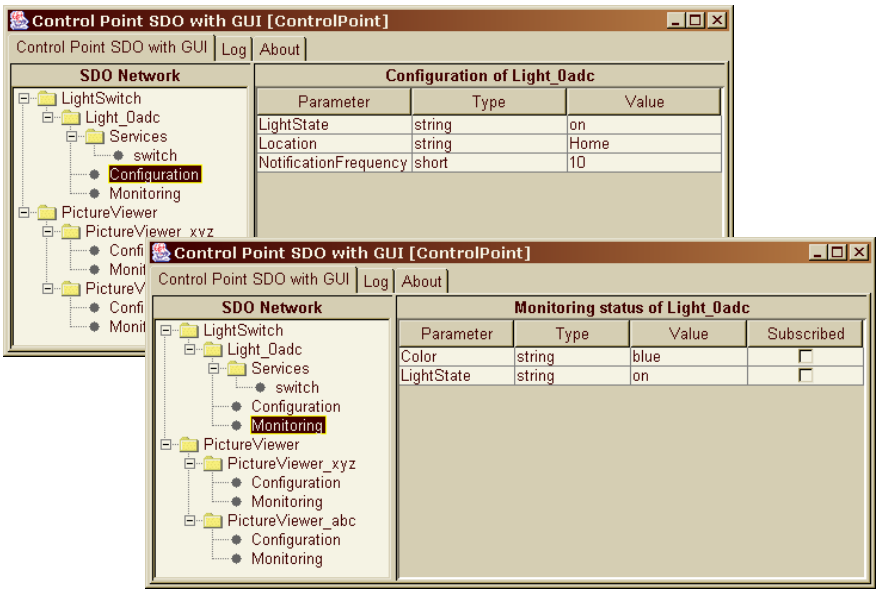


Figure 123: The configuration and monitoring panel

The SDO network tree is updated in real-time. Whenever a new SDO is announced in the network or an SDO either expires or leaves the network the tree is updated accordingly.

4.6 Cooperation inside the Open Profiling Framework

There are several kinds of events, which lead to activities in the proposed framework. For example, a Super Distributed Object can receive new information from the Sensing and Controlling Environment and is therefore sending it to the Context Server. On the other hand, a Context Interpreter can for instance process recent profiles of an individual and adapt machine created profiles at the Context Server for better personalization.

Since there is no restriction for the provision of new SDO components, activity can be the result of events from many different sources. Generally, these events can be assigned to one of the following classes:

- direct or indirect interaction between an individual and the I-centric communications system;
- interaction between SDOs and devices or applications of the real world;
- environmental changes detected by sensor devices; or
- automatic profile evaluation processes started by Context Interpreter or I-centric Service components.

The processing of these system activities is the subject of the following sections.

4.6.1 Component Cooperation

The central component of the framework is the Context Server, which is supported by Super Distributed Objects and Interpreter Objects like Context Interpreters or I-centric Services.

Figure 124 gives an overview of a typical activity inside the proposed framework. In the beginning, after launch of all components, Interpreting Components subscribe at the Context Server. Hence, the Context Server builds a list of all subscribed profile categories. The actual activity begins, when a Super Distributed Object receives an event through the underlying Sensing and Controlling Environment. The SDO component evaluates the event and decides, whether the event holds information that is worth sending it to the Context Server or if it is irrelevant. This first evaluation step is important because of the multitude of information an underlying real world device like a microphone can provide. The Context Server has to be protected against ‘information flooding’, which means that a huge amount of irrelevant information is sent that prevents it from the processing of important events.

If the event contains new and relevant information, it is sent by the Super Distributed Object component to the Context Server. The sent information is categorized and added to the appropriate profile (Context Profile or Relation Profile) as well as to the History Profile. Then, the Context Server evaluates if an interpreter object has subscribed the category of the just added data. If this is the case, the data is also sent to the subscribed component with use of the operation `NotifyChange`.

In the example, a Context Interpreter has subscribed the respective category. It is therefore notified about the new or changed data. The Context Interpreter might need additional information for the processing of this data, which is collected from the Context Server with the operation `GetData`. If the processing of the data results in new information, it is sent back to the Context Server where it is categorized and stored.

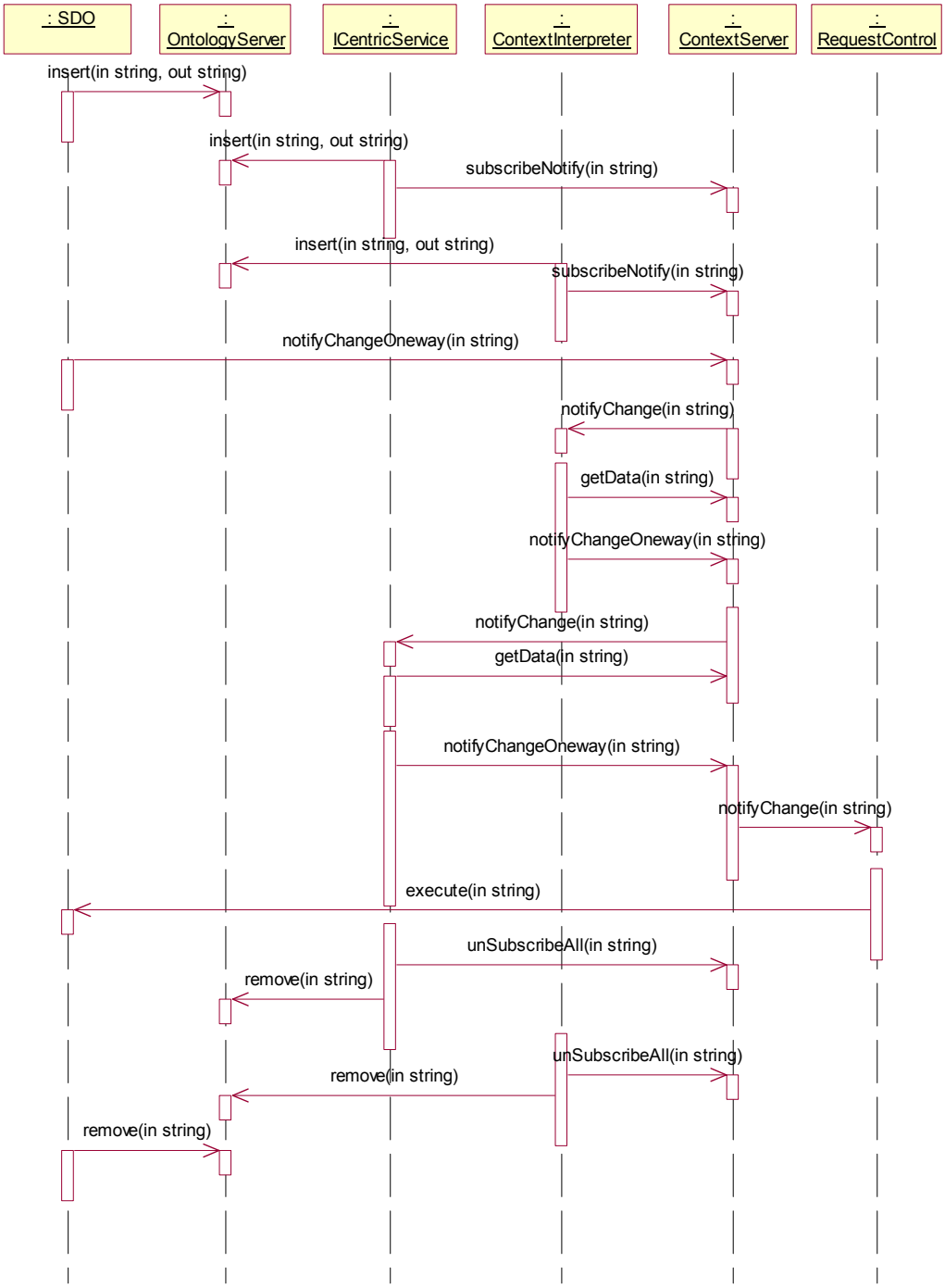


Figure 124: MSC: Framework Cooperation

An I-centric Service, which has subscribed information with that category, is notified. The I-centric Service again processes the data in the same way as the Context Interpreter. Only the result of the processing differs: The I-centric Service might decide that a certain action has to be performed now. It therefore uses the operation `Execute` of the SDO component that is responsible for access to the real world entity that can fulfill the action. Additionally, the I-centric Service sends information about this to be performed action or the result of the processing to the Context Server. Finally, the activated Super Distributed Object performs the action.

4.7 Demonstration Setup

To demonstrate the features of the developed system, a mobile demonstration scenario has been developed. The setup has been successfully demonstrates at several occasions (e.g. at OMG TCM in Helsinki in 2002). The SDOs (partially simulating physical devices only) involved in the mobile demonstration scenarios consist of: Brightness Sensor, Badge Sensor, Air condition (GUI), Jukebox (GUI), Picture Frame (GUI), Light (GUI), Dimmer (GUI).

This setup was chosen to simulate two rooms where users can roam between. An IR-Badge[35] is assigned to a certain user. IR-Tags[36] are assigned to certain rooms. In the concrete scenario one Badge is assigned to Stephan Steglich, the other one is assigned to Stefan Arbanowski. The IR-Tags are assigned to 'Room1' and 'Room2'. The users can move between the two locations. They will be recognized via their IR-Badges. The I-centric Services monitor the location of certain users and act accordingly. The services 'Customized Steglich' and 'Customized Arbanowski' control the status of Dimmers, Lights, Picture Viewers, and JukeBoxes. The personal preferences of the two users will be mapped to requests to control SDOs representing all these devices. Concurrent requests are handled by the service 'Request Control'. E.g., if Stefan Arbanowski is already in the room 'Room1' and Stephan Steglich follows him, the requests of Stephan Steglich will only be executed if they do not conflict with the request of Stefan Arbanowski. When Stefan Arbanowski leaves the room, the pending requests of Stephan Steglich will be applied to the SDOs immediately.

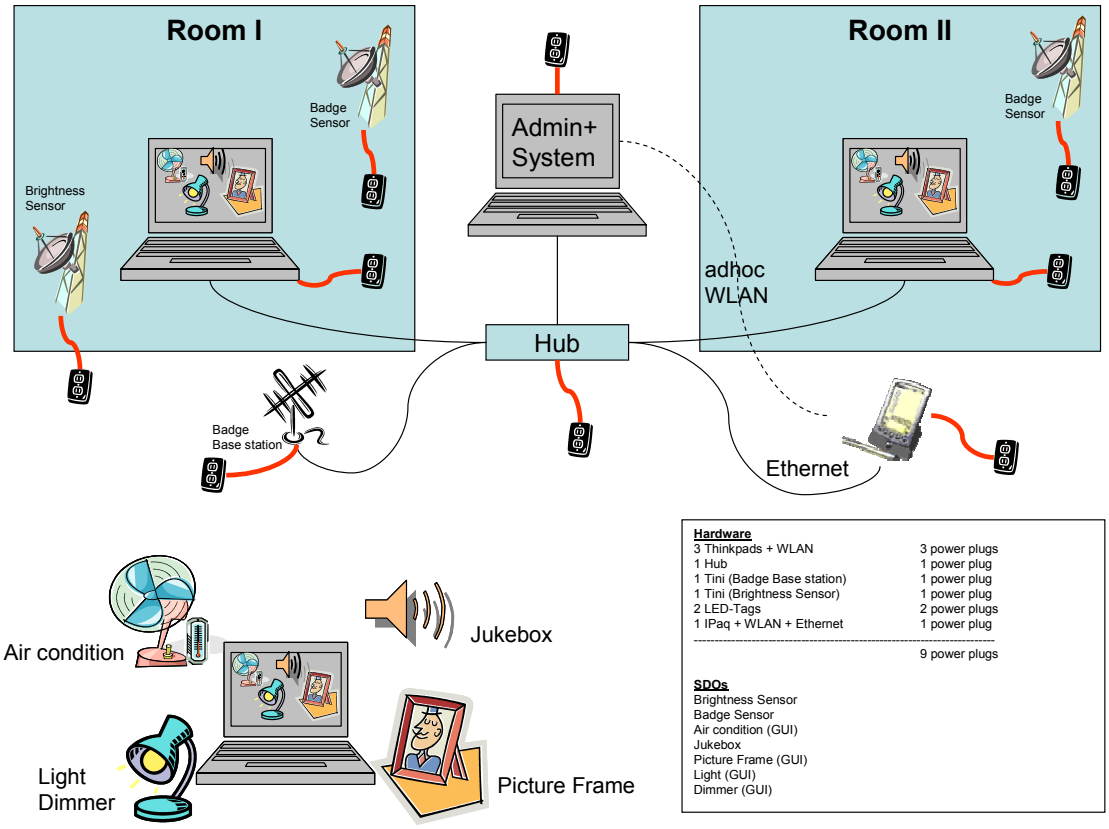


Figure 125: The Mobile SDO Demonstration Setup

Another service is 'Constant Brightness'. It controls the state of all available dimmers in assigned rooms. Therefore, a default brightness value is given to the service. If the brightness sensor detects a change of the brightness level, 'Constant Brightness' adjusts all Dimmers to get

[35] An IR-Badge is a small device equipped with an infrared emitter that sends out an ID frequently.

[36] An IR-Tag is a small device equipped with an infrared receiver that receives data sent by IR-Badges.

back to a preconfigured brightness value. In addition to that, ‘Constant Brightness’ takes user profiles for user preferences into account. If one user is a room, the configured preference of this user will be applied to all Dimmers. If there is more than one user, ‘Constant Brightness’ calculates the average of all preferences and controls all Dimmers accordingly. The ‘Constant Brightness’ Service can be subscribed and configured in the WWW-Interface. Each user can configure his personal preferences in an easy way.

Figure 126: Demo Desktop with Software SDOs

Figure 126 shows the demo setup of one computer representing the ‘Home’ location. Another computer represents the ‘Office’ location. The changes of state of the deployed SDOs can be observed via graphical user interfaces each SDO provides. Furthermore, the SDOs can be controlled directly via their user interfaces or the WWW-GUI.

4.7.1 Realized Scenarios

This section gives an overview about the service scenarios, which have been implemented beside the mobile demonstration, to prove the developed concepts. The three scenarios that have been implemented are:

- **automatic home control**:
  when an individual enters a room, several devices are configured to user‘s preferences
- **interactive home control**:
  the individual controls devices through a WWW-based room control application
- **home surveillance**:
  home is monitored and under specified conditions alarms are triggered

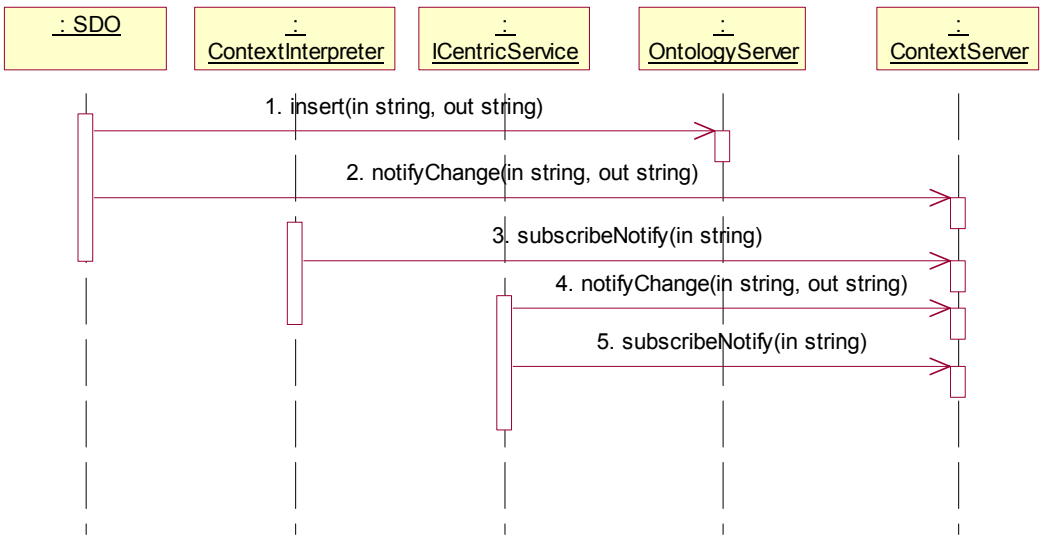


Figure 127: General registration sequences of SDO components

Figure 127 shows the general registration sequences of components that are performed at the beginning of each demonstration scenario. In each scenario, SDOs, Services, and ContextInterpreters register with the Ontology Server and with the Context Server subscribing to the relevant data if necessary. For simplification, these sequences are not shown in following message sequence charts describing the individual scenarios, but belong to each demonstration scenario.

4.7.1.1 Automatic Home Control

The main features are:

- Service Personalization – room is adjusted to personal preferences
- Automatic facility control – several SDOs are controlled in parallel

The Scenario:

- User one (Stefan Arbanowski) enters Room I.
- According to his Profile
    - the Light will be dimmed to 50% (This value can be specified easily using the Interactive Home Control User Interface.)
    - the Jukebox begins playing his favorite classic music
    - the Picture Frame will display his favorite painting(s)

This will be done by two different I-centric Services, ‘Customized Arbanowski’, and ‘My Light’. The specification of the second service is given as an example in the sections 7.4.1 (XML) and 7.4.2 (XSL).

- The ‘Customized Arbanowski’ service, which does not use personal profile information in the Context Server. It acts on the behalf of one specific user only and have the knowledge about the user’s preferences itself. If the user wants to have a different behavior, (for example, his preferences have changed), the service has to be adapted.
- The ‘My Light’ service uses personal preferences, which are stored in the Context Server. It can be subscribed by every user. It runs continuously. When no user is present in a room, it keeps the brightness level to one fixed value (e.g. 20%). If a user enters a room it adapts the brightness level to the value that is specified in the entering user’s personal profile (stored in the Context Server).

The scenario continued:

- User (Stefan Arbanowski) moves to Room II

- Room I returns to its former state
- Room II adjusts as before Room I (see above)
- User two (Stephan Steglich) enters Room II (now 2 persons are in this room at the same time). He has own, different preferences:
    - 100% light
    - his favorite pictures & music
    - air condition on
- The service negotiates between both users' preferences. Because user one is in the room already, only the air condition will be switched on and the light will be dimmed to 75%, which is average of 50% and 100%. The other SDOs are not changed.
- User one leaves Room II. According to the profile of Stephan Steglich, the room will be adjusted to
    - 100% light
    - his favorite pictures & music
- finally, user two leaves the room too. The room returns to original state:
    - light is dimmed to the default value
    - music playing stops
    - no pictures are shown anymore

Figure 128 shows the Message Sequence Chart of the Automatic Home Control scenario.

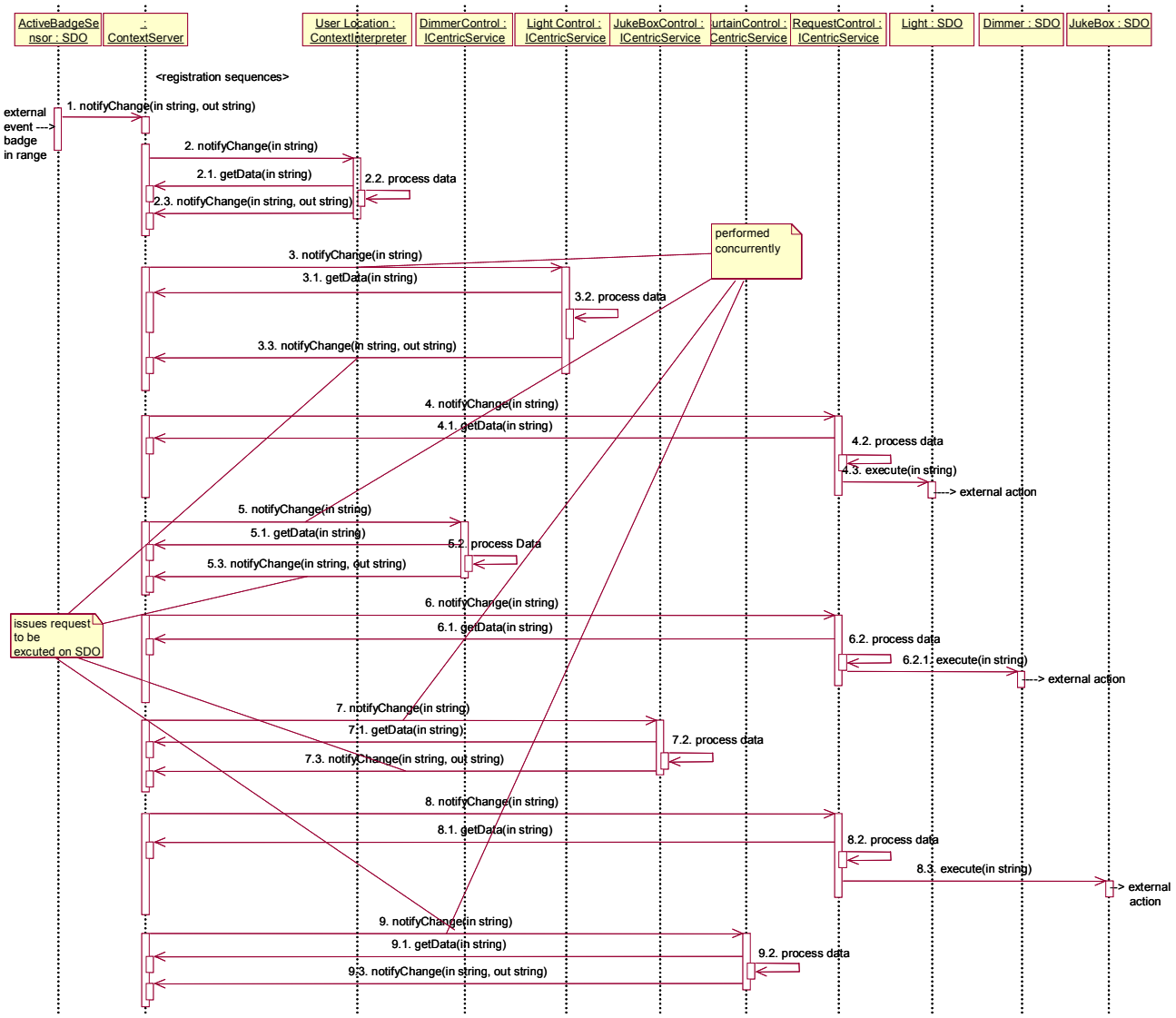


Figure 128: Automatic Home Control MSC

All information exchange is done via the Context Server (CS). The appearance of the user in the room is discovered by an ActiveBadgeSensor, which is acting as an SDO. The BadgeSensorSDO sends the sensed information (ID of a badge in range) to the CS where the LocationToRelation Context Interperter has subscribed exactly this kind of information.

Beside the notified dynamic data (badge ID seen), this Context Interpreter evaluates the static data (user-badge relation and location-badge sensor relation).

Then, the Context Interpreter sends the information which user is now residing in which room back to the Context Server. The Context Server now notifies a number of services, which have subscribed to this information (Customized Arbanowski, Customized Steglich, and Brightness Control).

They request (getData) all necessary information to process the service logic. If the services determine, that the state of one or more SDOs shall be changed according to the user preferences (e.g. a light shall be switched on), they issue an according request to the Context Server.

These requests are notified to a special service (RequestControl), which synchronizes all requests to be executed on SDOs. It remembers, which requests are already issued by services in order to avoid contrary controlling of SDOs (e.g. the light should not be switched off as long there is a user in the corresponding location, who prefers to have the light on). The RequestControl controls the SDOs directly calling the execute() method of the SDO interface, which is provided by each SDO.

4.7.1.2 Interactive Home Control

The main features are:

- interactive control of SDOs
- dynamic service discovery (available SDOs, offered service)
- user specific, personal description for SDOs
- interactive definition of service preferences

A user can discover dynamically what kind of devices or services he can control at a certain location. This discovery mechanism will be provided via a graphical user interface (WWW). The user accesses the webapplication with an ordinary WWW browser, which can be on a usual PC, on a webpad, or on a PDA.

The webapplication presents automatically all SDOs available in the location in which the user currently resides. Then he can directly control these devices. This scenario can be easily mapped to ‘wall mounted room control’ GUIs or to ‘remote home portal sites’ that enable the access and control of home facilities on the way. If there is no location information found for the user, he can manually choose a location, whose SDO he would like to control.

However, the presentations of individual SDOs are generated from the XML Schema-based ontology description. Because of the strict typing in XML Schema, there is an automatic generation of HTML code possible that allows the user to initiate requests to be executed on the SDOs.

he webapplication again converts the received HTTP requests in SDO requests that are then forwarded to the Context Server. In this way, the webapplication has not to know the existing SDO components. It uses SDOs in a generic way.

- User one accesses the room he is in via the WebAccess (see Figure 129)
    - Controls all devices (on, off, play, …)
    - Renames some devices to its personal preferences
    - Subscribes services and customizes services (e.g. set the preferred temperature for the temperature control service)
- User two does the same but uses a mobile PDA (see Figure 129):

Figure 130 shows the Message Sequence Chart describing the Interactive Home Control scenario. For this scenario, a JSP-based web application was developed. With it, the user can control all SDOs through a standard WWW-browser.

The web application (a servlet executed in a Tomcat webserver) generates a HTML presentation of the SDO functions interpreting the ontology description of the particular SDO. It interprets the XML Schema specification and transforms it into a suitable HTML presentation.

Figure 129: Personalized User Interfaces

Furthermore, the web application interprets certain data from the Context Server like location relations in order to present SDOs grouped according to their location. The web application considers user preferences stored in the Context Server in order to present SDOs with the user given names.

It allows also to subscribe to available services (such as Constant Brightness) and then to specify the personal preferences for these services. The web application provides suitable web pages for small handheld devices (with a reduced display of information, but providing the same functionality) as shown in Figure 129.

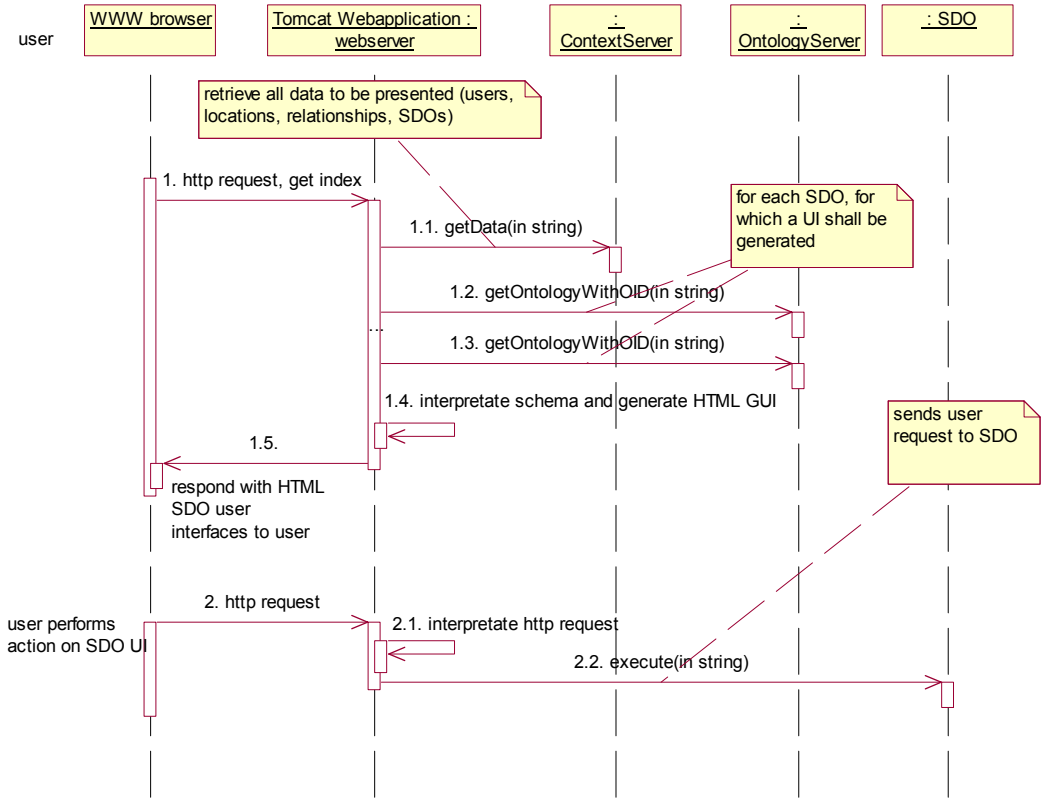


Figure 130: Interactive Home Control MSC

4.7.1.3 Home Surveillance

Several services can register with the Context Server to be informed about these events. The home surveillance scenario can utilize a lot of environmental sensors and services that react on events issued by sensors. To keep the scenario simple, only notifications and alarms will be triggered if a particular ActiveBadge is in motion and the owner of the according device is not in the same room.

This depicts a theft protection scenario, because the owner of device is notified immediately when the device is being moved. The owner can configure any number of actions to be done. In the demonstration scenario, the user has configured that he wants to receive an alarm notification on his mobile phone and that additionally a phone call with a pre-recorded announcement to a given number (i.e. belonging to the security service) is triggered.

This scenario demonstrates the continuous sensing of environmental data (observing many SDOs) and event-based reaction if specific events (sensed data) exceed pre-defined conditions (threshold values). Furthermore, it demonstrates that the user can flexibly specify what actions shall be triggered under which conditions.

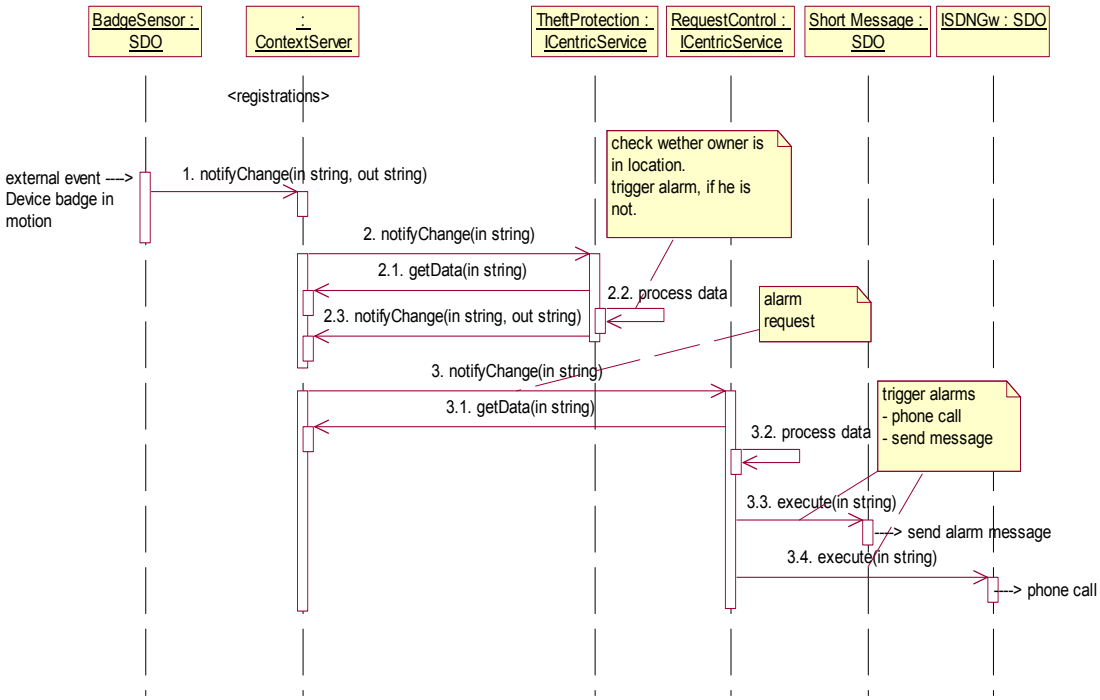


Figure 131: Home Surveillance MSC

Figure 131 shows the Message Sequence Chart describing the Home Surveillance scenario. In this scenario the user has configured, that if a certain device, to which an equipment badge is attached, is being moved while he is not residing in the same room, a message is sent to his mobile phone, and a phone call is triggered to a given number. The motion of the device is recognized by the BadgeSensorSDO, which receives a badge-in-motion event. The sensor sends this information to the Context Server, which notifies immediately the Theft Protection service about this. This service evaluates if the owner of device is residing in the same room (i.e. if there is an according room-user relation, which is created, when the user's badge is in range of the BadgeSensor in this room). If the user is not in the room, the service triggers the pre-defined actions (to send a Short Message to the user's GSM phone and to make a phone call to a given number and play back an announcement). In the scenario, the alarm state is canceled when the owner of the device enters the room, but this can be changed to any other events.

4.7.2 Realized I-centric Services

Each service consists of an XML and an XSL document. The XML document describes the service and it is used for the registration with the Context Server. The XSL document represents the logic of the service. It is used by an XSL-transformer to process the data that is retrieved from the Context Server after a notification. As shown in Figure 132 each service sends its description (the XML document) to the Context Server to register with the Context Server (1). Furthermore, each service subscribes to some data, which can be of static or dynamic nature (2). If the subscribed data has changed, the service will be notified immediately with the NotifyChange() method (3). The service can then obtain all relevant data from the Context Server using the getData() method (3.1). These data is then processed by an XSL-transformer using the service's XSL document (3.2). The result of the transformation is sent to the Context Server with the NotifyChange() method (3.3). The returned data can now cause the Context Server again to notify another service, which has subscribed to this data. The individual services, which are part of the demonstrated scenarios, are introduced more detailed in the following sections.

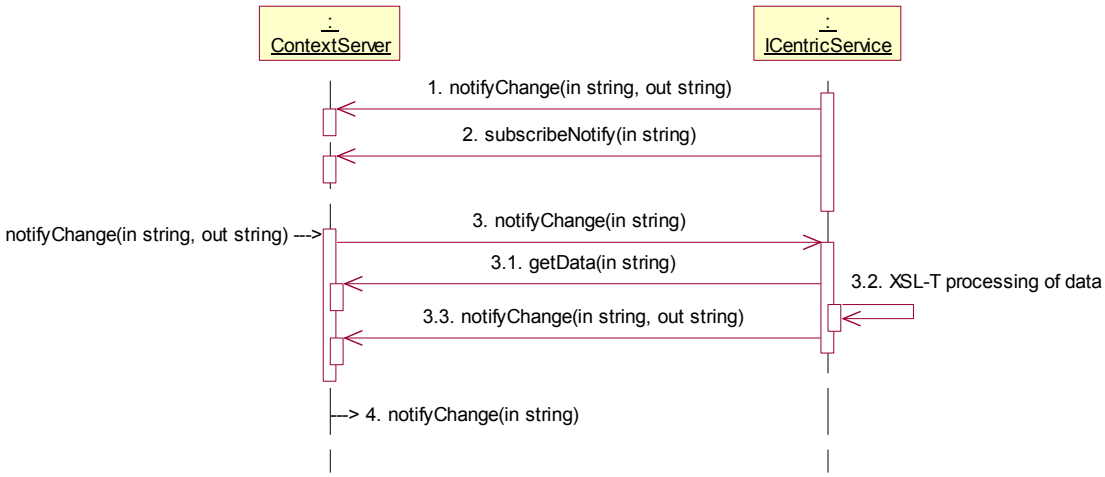


Figure 132: Interaction of Services with Context Server

4.7.2.1 Location Context Interpreter

The Location Context Interpreter subscribes to static relations BadgeID/actor and BadgeSensor/location, which are created by administrative processes. Furthermore, it subscribes to the data generated by BadgeSensor SDOs, which notify this data to the Context Server. With the notification by the Context Server, that this data has changed, the actor/location Context Interpreters knows the BadgeID(s) that are currently seen by a BadgeSensor. The XSL instructions evaluate this information together with the BadgeID/actor and BadgeSensor/location relations and calculate which user is residing in which location. It returns according actor/location relations back to the Context Server. If badges are not seen anymore by a BadgeSensorSDO, the actor/location Context Interpreter removes the according actor/location relation from the Context Server.

4.7.2.2 Theft Protection Service

The Theft Protection service subscribes to data provided by BadgeSensorSDOs, specifically to the equipment badges (i.e. all badges that have a relation to a device). Whenever there are events related to these equipment badges sent by a BadgeSensorSDO, the service is notified by the Context Server. The service then checks whether the device is in motion (this information is included in the notifications of the BadgeSensorSDOs) and if the owner of corresponding device is not in the same location. For this, the services analyzes the relations device-badgeID/device, actor/device (owner relation), and actor/location (where the owner currently resides).

If the equipment badge is in motion and the owner of the corresponding device is not in the same location, the service triggers alarms as specified in its XSL document. In the concrete scenario, it generates request for a Short Message SDO and for an ISDN gateway SDO. These requests are sent to the Context Server, which notifies the Request Control service, which sends the request to the appropriate SDOs (ShortMessageSDO and ISDNSDO). The ShortMessageSDO sends a predefined short text message to a given GSM telephone number as specified in the Theft Protection service XSL document. The ISDNSDO establishes a telephone call to specified telephone number and plays back a certain audio file (an alarm tone in the current demonstration scenario). The action, to be executed in case of alarm, can be changed, in the service's XSL document. For example, additional actions such as locking the door can be added easily. Telephone numbers to be called, text to be send, file to be played back can be changed respectively.

4.7.2.3 Customized User

This I-centric Service enables individual users to configure their environment. These services control in parallel the personal environment of the two users. The services are preconfigured with the personal preferences of the two users to control all SDOs that represent Lights, Dimmers, JukeBoxes, and Picture Viewers. Concurrent access to SDOs is controlled by the 'Request Control Service'. If the preference of one user is applied to an SDO, the service of the second user is not able to change the state of the SDO. The only possibility to control SDOs in this situation is to use direct access via the WWW-GUI. Otherwise, the first user has to leave the room. Then the preferences of the second user will be applied to the SDO environment automatically. The resulting scenario is that both users can move between different locations and their preferences will be applied to the surrounding environment.

4.7.2.4 Constant Brightness

'Constant Brightness Service' controls Dimmers according preconfigured user preferences. This service has to be subscribed and configured by users, which want their preferences of brightness levels applied to SDO environments automatically. Therefore, the WWW-GUI provides the possibility to subscribe and to configure services. In addition to that the 'Constant Brightness' is configured with a default brightness value. I.e., if no subscribed user is in a room, this default value is established by the service. This can be very useful for environments with changing levels of sunshine through the windows. Once configured, the room permanently stays at the same brightness level.

The consideration of user profiles by this service leads again to scenarios that are more flexible. Different users can have different preferences. 'Constant Brightness' averages the preferences of all users in a room and controls all dimmers accordingly. If there were three users in one room with the different preferences of brightness levels (e.g. 20%, 50%, 100%) the service would configure the room to a brightness level of 57%.

4.7.3 Realized SDOs

This section illustrates some of the implemented SDOs. The SDOs presented in this section have been selected because of their different nature.

4.7.3.1 Picture Viewer

This SDO controls a Picture Viewer that can show images that are stored at Web servers or local directories. Therefore, the ontology of Picture Viewer contains a schema specification for EXECUTE messages. Based on this specification, I-centric Services construct the instructions that are sent to the SDOs using the `execute()` method.

Constraints

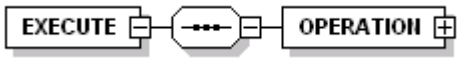
The I-centric Service environment requires additional information to control Picture Viewer SDO. For each SDO, a relation is required:

- picture viewer ↔ room.

Based on this relation, an I-Centric Service discovers all Picture Viewers at a certain location dynamically. This enables scenarios where personalized images can be displayed for certain users when they are nearby the Picture Viewer.

EXECUTE

The following table shows the XML Schema specification of the EXECUTE message. It is a passage extracted from the SDO ontology.

diagram	
children	**OPERATION**
source	<xs:element name="EXECUTE"> <xs:complexType> <xs:sequence> <xs:element name="OPERATION"> <xs:complexType> <xs:sequence> <xs:element name="PARAMETER"> <xs:complexType> <xs:attribute name="NAME"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="URL"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="VALUE"> <xs:simpleType> <xs:restriction base="xs:string"/> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="NAME"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="Show"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

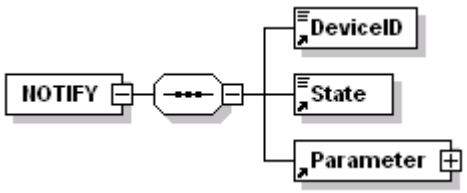
EXECUTE example

This example shows a typical EXECUTE-message that is executed by a Picture Viewer SDO. A certain light is addressed (`DeviceID`) that will be switched.

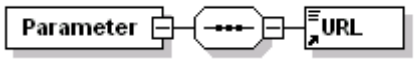
```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <OPERATION NAME="Show">
          <PARAMETER NAME="URL" VALUE="http://www.sdo.com/image3.jpg"/>
  </OPERATION>
</EXECUTE>
```

NOTIFY

The following table shows the XML Schema specification of the NOTIFY message. It is a passage extracted from the SDO ontology.

diagram	
children	**DeviceID State Parameter**
source	<xs:element name="NOTIFY"> <xs:complexType> <xs:sequence> <xs:element ref="DeviceID"/> <xs:element ref="State"/>

```
    <xs:element ref="Parameter"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
```

diagram	
children	URL
used by	element NOTIFY
source	<xs:element name="Parameter"> <xs:complexType> <xs:sequence> <xs:element ref="URL"/> </xs:sequence> </xs:complexType> </xs:element>

Example for NOTIFY

This example shows a typical NOTIFY-message that is sent by Picture Viewer SDOs. With such a message, this SDO reports about changes of its state (e.g. showing a certain picture).

```
<?xml version="1.0" encoding="UTF-8"?>
<NOTIFY xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <DeviceID>PictureViewerOffice</DeviceID>
  <State>Showing</State>
  <Parameter>
          <URL>http://www.sdo.com/picture.jpg</URL>
  </Parameter>
</NOTIFY>
```

4.7.3.2 Badge Sensor

This SDO controls sensors that detect infrared badges. The information about detected infrared badges is notified to the Context Server. Therefore, the ontology of Badge Sensors contains a schema specification for the NOTIFY messages. Based on this specification the Bade Sensor SDO constructs the messages and sends them to the Context Server using the `notifyChange()` method.

Constraints

The I-centric Service environment requires additional information to process information coming from the Badge Sensor SDO. Two relations are required:

- badge ↔ user,
- sensor ↔ room.

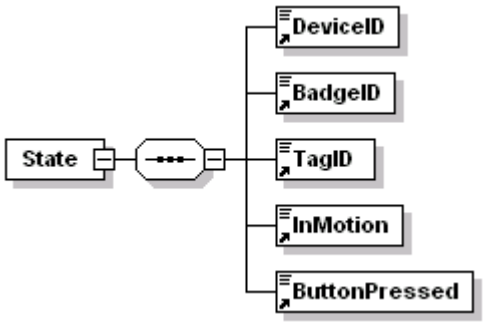
Based on these relations, an I-Centric Service can assign users to rooms automatically (i.e. create user ↔ location relation).

NOTIFY

The following table shows the XML Schema specification of the NOTIFY message. It is a passage extracted from the SDO ontology.

diagram	
children	State
source	<xs:element name="NOTIFY"> <xs:complexType> <xs:sequence> <xs:element ref="State"/> </xs:sequence>

	</xs:complexType> </xs:element>

diagram	
children	DeviceID BadgeID TagID InMotion ButtonPressed
used by	element NOTIFY
source	<xs:element name="State"> <xs:complexType> <xs:sequence> <xs:element ref="DeviceID"/> <xs:element ref="BadgeID"/> <xs:element ref="TagID"/> <xs:element ref="InMotion"/> <xs:element ref="ButtonPressed"/> </xs:sequence> </xs:complexType> </xs:element>

Example for NOTIFY

This example shows a typical NOTIFY-message that is sent by Badge Sensor SDOs. Interesting scenarios can be realized by interpreting the different parameters. E.g. for automatic processing the `InMotion` tag can be used, whereas for interactive scenarios the `ButtonPressed` tag is more interesting.

```
<?xml version="1.0" encoding="UTF-8"?>
<NOTIFY xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <State>
          <DeviceID>BadgeSensor06</DeviceID>
          <BadgeID>0f</BadgeID>
          <TagID>06</TagID>
          <InMotion>false</InMotion>
          <ButtonPressed>false</ButtonPressed>
  </State>
</NOTIFY>
```

4.7.3.3 Light

This SDO controls lights that can be switched *on* and *off*. Therefore, the ontology of Light contains a schema specification for EXECUTE messages. Based on this specification I-centric Service construct the instructions that are sent to the SDOs using the `execute()` method.

Constraints

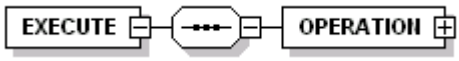
The I-centric Service environment requires additional information to control Light SDO'. For each SDO, a relation is required:

- light ↔ room.

Based on this relation, an I-Centric Service discovers all lights in a certain location dynamically.

EXECUTE

The following table shows the XML Schema specification of the EXECUTE message. It is a passage extracted from the SDO ontology.

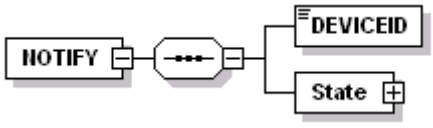
diagram	
children	**OPERATION**
source	<xs:element name="EXECUTE"> <xs:complexType> <xs:sequence> <xs:element name="OPERATION"> <xs:complexType> <xs:sequence> <xs:element name="PARAMETER"> <xs:complexType> <xs:attribute name="NAME"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="State"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="VALUE"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="on"/> <xs:enumeration value="off"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="NAME"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="ChangeState"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

EXECUTE example

This example shows a typical EXECUTE-message that is executed by a Light SDO. A certain light is addressed that will be switched.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <OPERATION NAME="ChangeState">
          <PARAMETER NAME="State" VALUE="on"/>
  </OPERATION>
</EXECUTE>
```

NOTIFY

diagram	
children	**DEVICEID State**
source	<xs:element name="NOTIFY"> <xs:complexType> <xs:sequence> <xs:element name="DEVICEID" type="xs:string"/> <xs:element name="State"> <xs:complexType> <xs:sequence> <xs:element name="State"> <xs:simpleType>

```
            <xs:restriction base="xs:string">
              <xs:enumeration value="on"/>
              <xs:enumeration value="off"/>
            </xs:restriction>
          </xs:simpleType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>
```

Example for NOTIFY

This example shows a typical NOTIFY-message that is sent by a Light SDO. With such a message, this SDO reports about changes of its state (e.g. on|off).

```
<?xml version="1.0" encoding="UTF-8"?>
<NOTIFY xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <DEVICEID>Light4008</DEVICEID>
  <State>
          <State>on</State>
  </State>
</NOTIFY>
```

4.7.3.4 Dimmer

This SDO controls lights that can be switched (*on/off)* and dimmed (*0-100%*). Therefore, the ontology of Dimmer contains a schema specification for EXECUTE messages. Based on this specification an I-centric Service constructs the instructions that are sent to the SDOs using the `execute()` method. Figure 133 shows the graphical user interface that has been implemented to control and to monitor the Dimmer SDO. All other SDOs have been implemented with similar user interfaces for simulation and demonstration purposes.

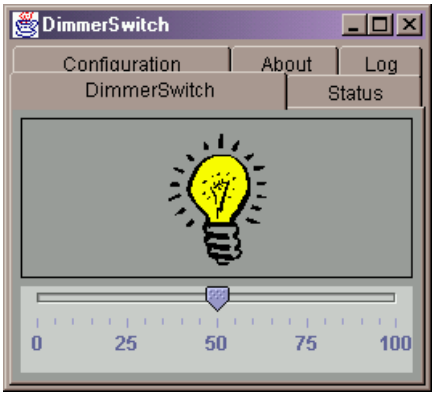


Figure 133: Dimmer GUI

The command switch(on/off) has been derived from the Light SDO and is expressed as a mapping in the ontology. Internally, the Dimmer is only able to switch the light by a given percentage. The Ontology Server provides a mapping functionality that maps ‘on’ to 100% and ‘off’ to 0%. In addition to that, the ontology of Dimmers contains a schema specification for NOTIFY messages. Based on this specification the Dimmer SDO constructs messages and sends them to the Context Server using the `notifyChange()` method.

Constraints

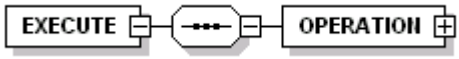
The I-centric Service environment requires additional information to control Dimmer SDOs. For each SDO, a relation is required:

- dimmer ↔ room.

Based on this relation, an I-Centric Service discovers all dimmers at a certain location dynamically.

EXECUTE

The following table shows the XML Schema specification of the EXECUTE message.

diagram	
children	**OPERATION**
source	<xs:element name="EXECUTE"> <xs:complexType> <xs:sequence> <xs:element name="OPERATION"> <xs:complexType> <xs:sequence> <xs:element name="PARAMETER"> <xs:complexType> <xs:attribute name="NAME"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="DimmValue"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="VALUE"> <xs:simpleType> <xs:restriction base="xs:int"> <xs:minInclusive value="0"/> <xs:maxInclusive value="100"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="NAME"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="Dimm"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

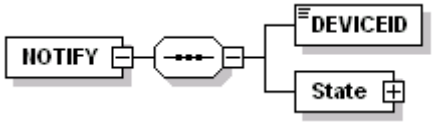
EXECUTE example

This example shows a typical EXECUTE-message that is interpreted by a Dimmer SDO.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <OPERATION NAME="Dimm">
        <PARAMETER NAME="DimmValue" VALUE="60"/>
  </OPERATION>
</EXECUTE>
```

NOTIFY

The following table shows the XML Schema specification of the NOTIFY message. It is a passage extracted from the SDO ontology.

diagram	
children	**DEVICEID** **State**
source	<xs:element name="NOTIFY"> <xs:complexType>

```
    <xs:sequence>
      <xs:element name="DEVICEID" type="xs:string"/>
      <xs:element name="State">
        <xs:complexType>
          <xs:sequence>
            <xs:element name="Dimmed">
              <xs:simpleType>
                <xs:restriction base="xs:int">
                  <xs:minInclusive value="0"/>
                  <xs:maxInclusive value="100"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
          </xs:sequence>
        </xs:complexType>
      </xs:element>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

NOTIFY example

This example shows a typical NOTIFY-message that is sent by Dimmer SDOs. With such a message, this SDO reports about changes of its state (e.g. dimmed to 30%).

```
<?xml version="1.0" encoding="UTF-8"?>
<NOTIFY xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <DEVICEID>Dimmer34</DEVICEID>
  <State>
          <Dimmed>40</Dimmed>
  </State>
</NOTIFY>
```

4.7.3.5 Jukebox

This SDO controls a JukeBox that can play personalized lists of MP3 songs. Therefore, the ontology of JukeBox contains a schema specification for EXECUTE messages. Based on this specification an I-centric Service constructs the instructions that are sent to the SDOs using the `execute()` method.

In addition to that, the ontology of JukeBoxes contains a schema specification for NOTIFY messages. Based on this specification the JukeBox SDO constructs messages and sends them to the Context Server using the `notifyChange()` method.

The graphical user interface of the Jukebox also provides direct control of all jukebox functions. Furthermore, there are special filters implemented in the jukebox. This can be used to parameterizes the jukebox with only one parameter (e.g. YEAR=’2000’). The jukebox searches all available mp3-files to find out which match the request. Then, the filtered list is played.

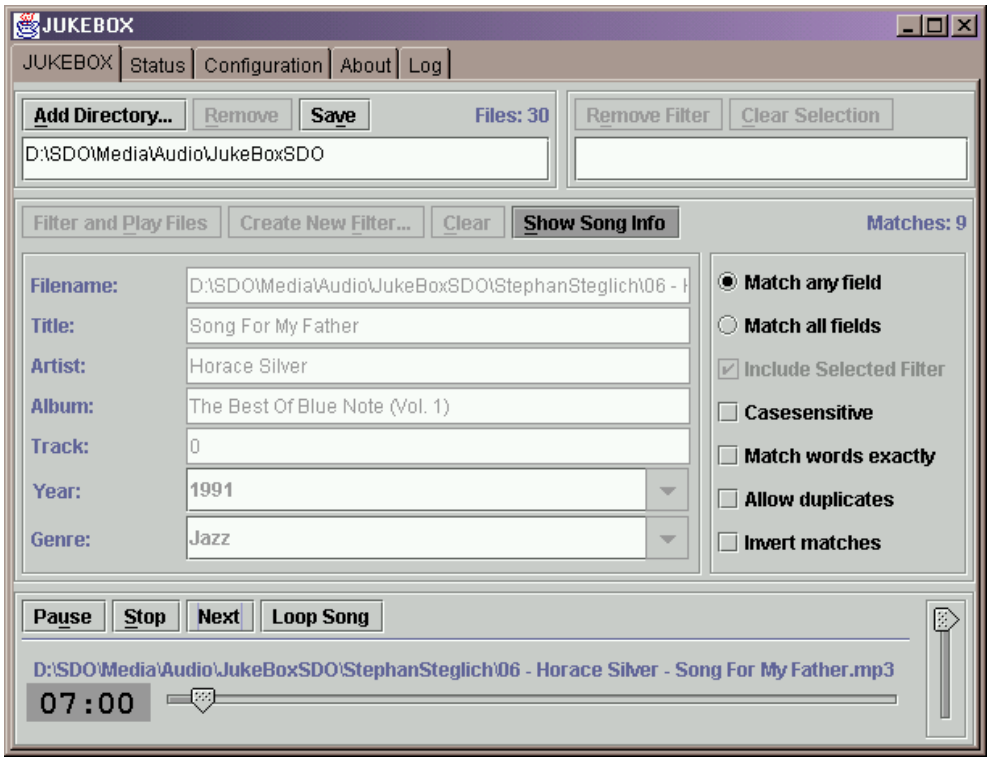


Figure 134: Jukebox GUI

Constraints

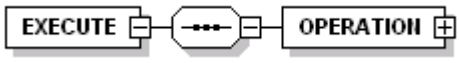
The I-centric Service environment requires additional information to control JukeBox SDOs. For each SDO, a relation is required:

- jukebox ↔ room.

Based on this relation, an I-Centric Service discovers all jukeboxes at a certain location dynamically.

EXECUTE

The following table shows the XML Schema specification of the EXECUTE message. It is a passage extracted from the SDO ontology.

diagram	
children	**OPERATION**
source	<xs:element name="EXECUTE"> <xs:complexType> <xs:sequence> <xs:element name="OPERATION"> <xs:complexType> <xs:sequence> <xs:element name="PARAMETER"> <xs:complexType> <xs:attribute name="NAME"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="artist"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="VALUE"> <xs:simpleType> <xs:restriction base="xs:string"/> </xs:simpleType>

```
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="PARAMETER">
    <xs:complexType>
      <xs:attribute name="NAME">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="year"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="VALUE">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="PARAMETER">
    <xs:complexType>
      <xs:attribute name="NAME">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="title"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="VALUE">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="PARAMETER">
    <xs:complexType>
      <xs:attribute name="NAME">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="file"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="VALUE">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="PARAMETER">
    <xs:complexType>
      <xs:attribute name="NAME">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="genre"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
      <xs:attribute name="VALUE">
        <xs:simpleType>
          <xs:restriction base="xs:string"/>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
  <xs:element name="PARAMETER">
    <xs:complexType>
      <xs:attribute name="NAME">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="album"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
```

```
            <xs:attribute name="VALUE">
              <xs:simpleType>
                <xs:restriction base="xs:string"/>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
        <xs:element name="PARAMETER">
          <xs:complexType>
            <xs:attribute name="NAME">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:enumeration value="volume"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:attribute>
            <xs:attribute name="VALUE">
              <xs:simpleType>
                <xs:restriction base="xs:string"/>
              </xs:simpleType>
            </xs:attribute>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
      <xs:attribute name="NAME">
        <xs:simpleType>
          <xs:restriction base="xs:string">
            <xs:enumeration value="play"/>
          </xs:restriction>
        </xs:simpleType>
      </xs:attribute>
    </xs:complexType>
  </xs:element>
 </xs:sequence>
</xs:complexType>
</xs:element>
```

EXECUTE example

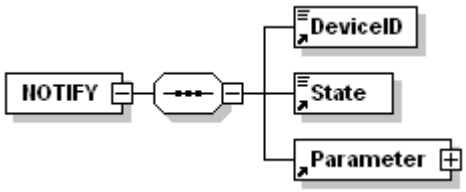
This example shows a typical EXECUTE-message that is interpreted by a JukeBox SDO.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <OPERATION NAME="play">
          <PARAMETER NAME="artist" VALUE="Horace Silver"/>
          <PARAMETER NAME="title" VALUE="Song For My Father"/>
          <PARAMETER NAME="file" VALUE="D:\SDO\Media\Audio\ForMyFather.mp3"/>
          <PARAMETER NAME="genre" VALUE="Jazz"/>
          <PARAMETER NAME="album" VALUE="The Best Of Blue Note (Vol. 1)"/>
          <PARAMETER NAME="volume" VALUE="80"/>
  </OPERATION>
</EXECUTE>
```

NOTIFY

The following table shows the XML Schema specification of the NOTIFY message. It is a passage extracted from the SDO ontology.

diagram



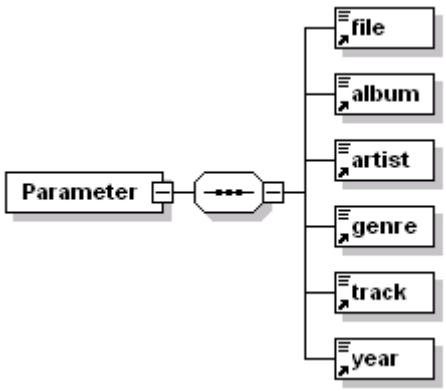
children **DeviceID** **State** **Parameter**

source

```
<xs:element name="NOTIFY">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="DeviceID"/>
      <xs:element ref="State"/>
      <xs:element ref="Parameter"/>
    </xs:sequence>
```

```
  </xs:complexType>
</xs:element>
```

diagram



children **file album artist genre track year**

used by element **NOTIFY**

source

```
<xs:element name="Parameter">
  <xs:complexType>
    <xs:sequence>
      <xs:element ref="file"/>
      <xs:element ref="album"/>
      <xs:element ref="artist"/>
      <xs:element ref="genre"/>
      <xs:element ref="track"/>
      <xs:element ref="year"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
```

NOTIFY example

This example shows a typical NOTIFY-message that are sent by JukeBox SDOs. With such a message, this SDO reports about changes of its state (e.g. state ‘play’).

```
<?xml version="1.0" encoding="UTF-8"?>
<NOTIFY xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <DeviceID>JUKEBOXHOME</DeviceID>
  <State>play</State>
  <Parameter>
  <file>D:\Projects\Media\StefanArbanowski\SymphonieFantastique.mp3</file>
          <album>Why</album>
          <artist/>
          <genre>Pop</genre>
          <track>32</track>
          <year>2000</year>
  </Parameter>
</NOTIFY>
```

4.7.3.6 Air-condition

This SDO controls an AirConditioning that can be switched *on* and *off*. Therefore, the ontology of AirConditioning contains a schema specification for EXECUTE messages. Based on this specification an I-centric Service constructs the instructions that are sent to the SDOs using the `execute()` method.

In addition to that, the ontology of AirConditioning SDOs contains a schema specification for NOTIFY messages. Based on this specification the AirConditioning SDO constructs messages and sends them to the Context Server using the `notifyChange()` method.

Constraints

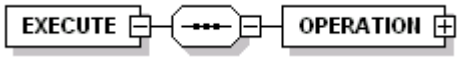
The I-centric Service environment requires additional information to control AirConditioning SDOs. For each SDO, a relation is required:

- airconditioning ↔ room.

Based on this relation, an I-Centric Service discovers all airconditioning SDOs at a certain location dynamically.

EXECUTE

The following table shows the XML Schema specification of the EXECUTE message. It is a passage extracted from the SDO ontology.

diagram	
children	**OPERATION**
source	<xs:element name="EXECUTE"> <xs:complexType> <xs:sequence> <xs:element name="OPERATION"> <xs:complexType> <xs:sequence> <xs:element name="PARAMETER"> <xs:complexType> <xs:attribute name="NAME"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="State"/> </xs:restriction> </xs:simpleType> </xs:attribute> <xs:attribute name="VALUE"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="on"/> <xs:enumeration value="off"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element> </xs:sequence> <xs:attribute name="NAME"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="ChangeState"/> </xs:restriction> </xs:simpleType> </xs:attribute> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

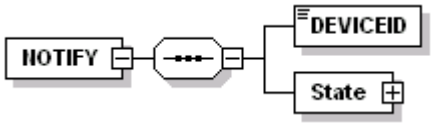
EXECUTE example

This example shows a typical EXECUTE-message that is executed by a AirConditioning SDO.

```
<?xml version="1.0" encoding="UTF-8"?>
<EXECUTE xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <OPERATION NAME="ChangeState">
          <PARAMETER NAME="State" VALUE="on"/>
  </OPERATION>
</EXECUTE>
```

NOTIFY

The following table shows the XML Schema specification of the NOTIFY message. It is a passage extracted from the SDO ontology.

diagram	
children	DEVICEID State
source	<xs:element name="NOTIFY"> <xs:complexType> <xs:sequence> <xs:element name="DEVICEID" type="xs:string"/> <xs:element name="State"> <xs:complexType> <xs:sequence> <xs:element name="State"> <xs:simpleType> <xs:restriction base="xs:string"> <xs:enumeration value="on"/> <xs:enumeration value="off"/> </xs:restriction> </xs:simpleType> </xs:element> </xs:sequence> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

NOTIFY example

This example shows a typical NOTIFY-message that are sent by AirConditioning SDOs. With such a message, this SDO reports about changes of its state (e.g. switched on).

```
<?xml version="1.0" encoding="UTF-8"?>
<NOTIFY xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <DEVICEID>Aircon723</DEVICEID>
  <State>
          <State>on</State>
  </State>
</NOTIFY>
```

4.7.3.7 Brightness Sensor

This SDO controls sensors that measure the level of brightness. The measured value is notified to the Context Server. Therefore, the ontology of Brightness Sensors contains a schema specification for NOTIFY messages. Based on this specification the Brightness Sensor SDO constructs the messages and sends them to the Context Server using the `notifyChange()` method.

Constraints

The I-centric Service environment requires additional information to process information coming from the Brightness Sensor SDO. For each SDO a relation is required:

- brightness sensor ↔ room.

Based on this relations, an I-Centric Service can assign a level of brightness to rooms automatically.

NOTIFY

The following table shows the XML Schema specification of the NOTIFY message. It is a passage extracted from the SDO ontology.

diagram	
children	**State**
source	<xs:element name="NOTIFY"> <xs:complexType> <xs:sequence> <xs:element name="State"> <xs:complexType> <xs:sequence> <xs:element name="DeviceID"> <xs:simpleType> <xs:restriction base="xs:string"/> </xs:simpleType> </xs:element> <xs:element name="Brightness"> <xs:simpleType> <xs:restriction base="xs:float"> <xs:minInclusive value="0.0"/> <xs:maxInclusive value="100.0"/> </xs:restriction> </xs:simpleType> </xs:element> </xs:sequence> </xs:complexType> </xs:element> </xs:sequence> </xs:complexType> </xs:element>

NOTIFY example

This example shows a typical NOTIFY-message that is sent by Brightness Sensor SDOs. With such a message, this SDO reports about the level of brightness in their environment.

```
<?xml version="1.0" encoding="UTF-8"?>
<NOTIFY xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <State>
          <DeviceID>BrightnessSensor</DeviceID>
          <Brightness>59.320004</Brightness>
  </State>
</NOTIFY>
```

4.8 Summary

This chapter introduced the implementation, which is based on the architectural framework introduced in chapter 3. The Open Profiling Framework and the Super Distributed Objects are discussed in detail. Started with a complete description of all implemented components, this chapter investigates the component cooperation inside the realized system, the necessary maintenance tools, the developed scenarios, I-centric Services, and SDOs.

5 Summary

This chapter concludes this thesis. It recalls the ideas, tasks, and results of the work undertaken in the area of I-centric communications systems. Since this thesis is embedded in several national and international research projects, the implications and possible future enhancements are discussed in the following.

5.1 Conclusion

Based on the introduced **Vision of I-centric Communications**, this thesis has presented a way to provide ambient-aware, personalized, and adaptive communication services. An appropriate I-centric communications system has been built. The system provides I-centric Services based on objects that characterize the individual communication space.

The design follows a top-down approach, analyzing human's communication behavior, instead of taking decisions based on the technology available. An Open Profiling Framework is proposed that integrates ambient-awareness, service personalization, and adaptability of services. Super Distributed Objects are introduced as enabling technology to model and implement the objects that fulfill individual user wishes. A mechanism is proposed to describe the business logic of I-centric Services in a generic way. This mechanism has been successfully used to novel user initiated service creation and deployment.

The **Open Profiling Framework** has shown the possibility to realize service creation, service deployments, service processing, and object interaction based on generic descriptions that describe both syntax and semantic of service's functionality (ontology).

The concept of **Super Distributed Objects** enables coordinated interaction among massive numbers of autonomous software components. These software components can act in an ad-hoc and peer-to-peer manner to fulfill specific demands caused by individual's preferences. Super Distributed Objects have been used to model and to implement the objects of individual communication spaces.

The Open Profiling Framework has been implemented on top of Super Distributed Objects. The implementation has proven that an environment based on Super Distributed Objects fulfills the requirements of **I-centric Services**.

All discussed approaches have been realized applying state-of-the-art technologies. Even doubts, regarding the performance of XML/XSL based technologies, could not confirmed to be true after analyzing the runtime behavior of the I-centric communications system.

5.2 Outlook

During the design and implementation phase of the I-centric communications system, some further research tasks have already been identified. The following sections give an overview about these research tasks by relating them to the Open Profiling System and the Super Distributed Objects. Finally, the relevance of I-centric Communications for the 3G*b* systems is discussed.

5.2.1 Open Profiling Framework

The Open Profiling Framework provides the semantic backbone for I-centric communications systems. During the design and implementation phase of the I-centric communications system, some possible enhancements have already been identified.

5.2.1.1 Security

The current implementation offers quite powerful query mechanisms to collect information about individual communication spaces including available objects, preferences, ambient infor-

mation, and service usage. Nevertheless, this information is measured partially as sensitive information. Permissions and prohibitions to access such kind of information have to be considered. Future research activities should analyze the possibilities to grant partial profile access, based on the roles, policies, and scope, a querying instance has.

5.2.1.2 Federation between I-centric Communications Systems

The Context Server has been introduced as central component of the Open Profiling Framework, storing information about all objects, preferences, and ambient information. Beside the fact, that the centralization of this functionality can be the bottleneck for large-scale I-centric systems, the focus of this work has been on a single instance. Nevertheless, the interworking between distributed instances of Context Servers could enhance the applicability of I-centric communications systems for scenarios running across different organizational boundaries or administrative domains. Future research activities should analyze the use of concepts that have already been developed in the area of service roaming, inter-domain federation, and inter-domain management. Two different approaches for interconnected Context Servers have already been identified, which are introduced below.

Interconnected Context Servers maintaining Shared Profiles

One approach that should be further analyzed uses interconnected Context Servers only for load balancing and performance optimization. Profiles have to be replicated and synchronized permanently between different Context Servers. The main problem to be researched is a distributed synchronization mechanism that ensures the correctness of all managed data.

Interconnected Context Servers maintaining Distributed Profiles

The second approach to be analyzed follows the idea to split profiles between different Context Servers. In this approach, load balancing is only a secondary issue. The distribution of parts of profiles between several Context Servers can help to ensure security and privacy issues. Sensitive parts of profiles can be managed be ‘trusted’ servers, whereas uncritical parts can be stored in ‘public’ servers. The main problem in this approach is the on demand assembling of needed profile data, which is then distributed among several instances of Context Servers.

5.2.1.3 Automatic Service Creation

Section 3.2.7.2 has introduced a first approach to generate I-centric Services without direct involvement of an individual. The approach follows the idea of identifying patterns of frequent service usage in the history information an I-centric communications system provides. Nevertheless, the introduced algorithm needs domain specific knowledge to perform its task.

Future research activities should analyze whether it is possible to generate new I-centric Services without any additional knowledge except that which is already provided by the history information of the system. Learning mechanisms, from the areas of Artificial Intelligence or Knowledge Based Systems, should be analyzed to apply them to the automatic creation of I-centric services.

5.2.2 Super Distributed Objects

5.2.2.1 Security

The Super Distributed Object (SDO) specification provides interfaces that focus on SDOs from two different perspectives. The first one aims for providing a usage interface that suits the functional requirements of I-centric communications. The second one focuses on the runtime-behavior and system management. Both types of interfaces do not explicitly address security aspects. Future research activities should investigate, whether it is possible to augment both

interface types with security mechanism that orientate on the roles, tasks, and policies, which are to be assigned to I-centric communications systems.

5.2.2.2 Organizational Model

The specification of Super Distributed Objects so far, addresses an SDO as autonomous entity without any static relationships to any other SDO or application. In real world scenarios, where ownership issues or organizational boundaries play an important role, this might be insufficient. The implemented system solves these issues through the Open Profiling Framework on top of the SDOs. However, specific application domains might request the embedment of organizational models inside the SDO middleware. This is also reflected by the SDO RFP [SDO-RFP] of OMG's SDO DSIG, where an organizational model for SDO has been requested as *optional* enhancement of the SDO environment. Future research activities should identify how organizational models can be integrated an SDO infrastructure without causing administrative overhead and by keeping the autonomous characteristic of a single SDO.

5.2.3 I-centric Communications for 3Gb Systems

The multitude of new technologies and services foreseen for 3G*b* systems (like new wired and wireless access technologies, new application protocols and end-devices, and value-added services trough service orchestration) seems to be a perfect playground for I-centric communications systems.

I-centric Services provide the orchestration of services running in different application domains inherently. A fast and easy introduction of new services based on existing ones is possible.

Available resources can be wrapped by SDO concepts to be integrated in individual communication spaces. Personalization, ambient-awareness, and adaptability enable the ubiquitous availability of end-user services according individual's preferences and current environments.

The vision of I-centric communications has already been introduced to the forum preparing and steering the developments towards 3G*b* systems (WWRF). It has been adopted by the WWRF and builds the basis for the definition of service architectures for 3G*b* systems, which are to be deployed around 2010.

6 References

6.1 Literature

[Abo98] Abowd, G.; Atkeson, C.; Brotherton, J.; Enqvist, T.; Gulley, P.; LeMon, J.: Investigating the Capture, Integration and Access Problem of Ubiquitous Computing in an Educational Setting, Proc. of CHI '98 Los Angeles, CA (April 18-23, 1998), pp. 440-447.

[Abo99] Abowd, G. D.; Dey, A. K. et. al.: Towards a Better Understanding of context and Context-Awareness. First International Symposium on Handheld and Ubiquitous Computing, HUC'99, Karlsruhe, Germany, 27-29 Sept., 1999

[Arb96] Arbanowski, St.: Generic Description of Telecommunication Services and Dynamic Resource Selection in Intelligent Communication Environments. - Diplomarbeit, Technical University of Berlin, Institute for Open Communications Systems (OKS), 1996

[Arb97] Pfeifer, T; Arbanowski, St; Popescu-Zeletin, R.: *Resource Selection in Heterogeneous Communication Environments using the Teleservice Descriptor*. 4th COST 237 Workshop, Lisboa, Portugal, December 15-19, 1997, pp. 132-153, ISBN 3-540-63935-7

[Arb98] Arbanowski, St; Breugst, M; Busse, I; Magedanz, T.: *Impact of Standard Mobile Agent Technology on Telecommunications*. 5th Conference on Computer Communications, AFRICOM-CCDC'98, Tunis, Tunisia, October 20-22, 1998, pp. 189-203

[Arb99a] van der Meer, S; Arbanowski, St: Magedanz, T: *An Approach for a 4th Generation Messaging System*. The 4th International Symposium on Autonomous Decentralized Systems, ISADS'99, Tokyo, Japan, March 21-23, 1999, pp. 158-167, ISBN 0-7695-0137-0

[Arb99b] Arbanowski, St; van der Meer, S: *Service Personalization for Unified Messaging Systems.* The 4th IEEE Symposium on Computers and Communications, ISCC'99, Red Sea, Egypt, July 6-8, 1999, pp. 156-163, ISBN 0-7695-0250-4

[Arb99c] van der Meer, S; Arbanowski, St: Popescu-Zeletin, R: *A Platform for Environment-Aware Applications.* 1st International Symposium on Handheld and Ubiquitous Computing, HUC'99, Karlsruhe, Germany, September 27-29, 1999, pp. 368-370, ISBN 3-540-66550-1

[Arb99d] van der Meer, S; Arbanowski, St: Popescu-Zeletin, R: *Environment-aware Applications: Integrating Mobile Communications and Ubiquitous Computing.* Proc. of 7th Annual International conference on Advances in Communication and Control, COMCON 7, Athens, Greece, June28 –July 2, 1999, p415-428, ISBN 0-911575-75-8

[Arb00a] van der Meer, S; Arbanowski, S; Steglich, St; Popescu-Zeletin, R: *The Human Communication Space Towards I-centric Communications.* Workshop: The What, Who, Where, When, Why and How of Context-Awareness, ACM Conference on Human Factors in Computing Systems, CHI2000, The Hague, The Netherlands, April 1-6, 2000

[Arb00b] van der Meer, S; Arbanowski, St; Steglich, St: *Flexible Control of Media Gateways for Service Adaptation.* Proc of the 5th IEEE Intelligent Network Workshop, Cap Town, South Africa, May 07-11, 2000, ISBN 0-7803-6317-5

[Arb00c] Arbanowski, St; van der Meer, S: Popescu-Zeletin, R: *I-centric Services in the Area of Telecommunication 'The I-Talk Service'.* Proc. of the 6th IFIP TC6/WG6.7 Conference on Intelligence in Networks, SmartNet 2000, Vienna, Austria, September 18-22, 2000, pp. 499-508, ISBN 0-7923-7932-2

[Arb00d] van der Meer, S; Arbanowski, St: *Service Interoperability through advanced Media Gateways.* Proc. of the 6th IFIP TC6/WG6.7 Conference on Intelligence in Networks, SmartNet 2000, Vienna, Austria, September 18-22, 2000, pp. 583-595, ISBN 0-7923-7932-2

[Arb00e] Arbanowski, St; Waterstrat, H; van der Meer, S; Popescu-Zeletin, R: *Open Profiling for Ubiquitous Computing.* Proc. of the 1st Workshop on Ubiquitous Computing, PACT 2000, Philadelphia, PA, October 15–19, 2000, ISBN 3-86009-191-3

[Arb00f] Arbanowski, St: *Open Profiling for I-centric Computing.* Proc. of the annual ACM Conference for the Computer-Human Interaction Special Interest Group (CHISIG) OZCHI 2000, Sydney, Australia, December 4-8, 2000

[Arb00g] van der Meer, S; Arbanowski, St: *Flexible Media and Content Adaptation for Communication Systems.* Proc. of the IEEE Conference on Protocols for Multimedia Systems, PROMS 2000, Cracow, Poland, October 22-25, 2000, pp. 461-477, ISBN 83-88309-05-6

[Arb01a] Arbanowski, St.; van der Meer, S.; Steglich, St.; Popescu-Zeletin, R.: *The Human Communication Space: Towards I-centric Communications*. Volume 5, Personal and Ubiquitous Computing, Issue 1, pp. 34-37, ISSN 1617-4909

[Arb01b] van der Meer, S; Arbanowski, St; Steglich, St: User-Centric Communications. Proc. of the IEEE ICT 2001 – IEEE International Conference on Telecommunications, Romania, 2001, Volume 4, pp. 452-444, ISBN 973-99995-3-0

[Arb01c] van der Meer, S., Arbanowski, St.: From Unified Messaging towards I-centric Services for the Virtual Home Environment. Proc. of the 6th IEEE Intelligent Network Workshop, IN2001, Boston, MA, USA, May 6-9, 2001, ISBN 0-7803-7047-3

[Arb01d] Arbanowski, St., Steglich, S.: Profiling Contexual Information. IEEE International Conference on Parallel Architectures and Compiling Techniques – Proc. of the Workshop on Ubiquitous Computing and Communication, Barcelona, 2001

[Arb01e] Arbanowski, St.; Steglich, S.: Profile Information based Service Creation. Proc. of the 4th Asia-Pacific Symposium on Information and Telecommunication Technologies, Tribhuvan University, Kathmandu, Nepal, 2001

[Arb01f] Strick, L.; Arbanowski, St.: Super Distributed Objects - A Concept for Personalized Services Portability Across Network and Terminal. Proc. of the 4th Asia-Pacific Symposium on Information and Telecommunication Technologies, Tribhuvan University, Kathmandu, Nepal, 2001

[Arb01g] Arbanowski, St.; van der Meer, S.; Steglich, St.; Popescu-Zeletin, R.: I-centric Communications. Informatik - Forschung und Entwicklung, Organ des Fachbereichs 2 der Gesellschaft für Informatik e.V. (GI), Springer-Verlag, Band 16, Heft 4, S. 225-232, 2001, ISSN 0178-3564 IFENEI

[Arb02a] Arbanowski, St.: 3G*b* Business Models: Technical Aspects. Proc. of the Symposium on Business Models for Innovative Mobile Services, Delft, Netherlands, November 15-16th

[Arb02b] Popescu-Zeletin, R.; Arbanowski, St.: I-centric Service Architectures for B3G. Proc. of the International Forum on Future Mobile Telecommunications & China-EU Post Conference on Beyond 3G, Beijing, China, November 20-22, 2002

[Arb03a] Popescu-Zeletin, R.; Arbanowski, St.; Fikouras, I.; Gasbarrone, G.; Gebler, M.; Henning, H.; van Kranenburg, H.; Portschy, H.; Postmann, E.; Raatikainen, K.: Service Architectures for the Wireless World. Computer Communications, Vol. 26, No. 1, January 2003, pp. 19 - 25, invited special issues, ISSN 0140-3664

[Arb03b] Arbanowski, St.; Lipka, M.; Mössner, K.; Ott K.; Pabst R.; Pulli P.; Schieder, A.; Uusitalo, M. A.: The WSI Reference Model for the Wireless World. Proc. of the 21st IST summit on mobile and wireless communications, Aveiro, Portugal, June 15-18, 2003, Volume 2, pp. 613-617, ISBN 972-98368-7

[Arb03c] Steglich, St., Vaidya, R. N., Gimpeliovskaja, O., Arbanowski, St., Popescu-Zeletin, R., Sameshima, Sh., Kawano, K.: I-Centric Services Based on Super Distributed Objects. ISADS 2003, Proc. of the 6th International Symposium on Autonomous Decentralized Systems, Pisa, Italy, April 9-11, 2003, pp. 232-239, ISBN 0-7695-1876-1

[Arb03d] Radusch, I., Arbanowski, St., Steglich, St., Popescu-Zeletin, R.: I-Centric Services based on Super Distributed Objects. Med-Hoc NET 2003 Workshop, Mahdia, Tunesia, March 26-27, 2003

[Arb03e] Arbanowski, St., Steglich, St.: Service Architectures for 3G and Beyond.
SICE Annual Conference 2003, Fukui, Japan, Augut 4-6, 2003

[Arb03f] Arbanowski, St.; Steglich, St.; Popescu-Zeletin, R.: Super Distributed Objects – an execution environment for I-centric Services. Proc. of the 9-th IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 2003F), Capri Island, Italy, October, 1-3, 2003

[Base02] Basekow, R.: Ontology Server for an I-Centric communications system. - Diplomarbeit, Technical University of Berlin, Institute for Open Communications Systems (OKS), 2002

[Bern] Advani, A.; Tu, S.; Musen, M.: Domain Modeling with Integrated Ontologies: Principles for Reconciliation and Reuse, Section on Medical Informatics, Stanford University School of Medicine

Stanford University, Stanford, CA 94305-5479

[Bre98] Breugst, M., Magedanz, M.: On the Usage of Standard Mobile Agent Platforms in Telecommunication Environments. 5th ACTS IS&N Conference, Antwerp, Belgium, May 25-28, 1998

[Borgo] Gangemi, A., Guarino, N., Masolo, C., and Oltramari, A.: Understanding top-level ontological distinctions, Proc. of IJCAI 2001 workshop on Ontologies and Information Sharing

[Brow2k] Brown, P.;Jones, G.: Workshop on Context-awareness: Position Paper/Extended Abstract. CHI2000 Workshop #11, The Hague, Netherlands, 01-06 April, 2000

[Cha01] Cha, S.: Apriori Algorithm for Sub-category Classification Analysis. Proc. of the 6th International Conference on Document Analysis and Recognition (ICDAR '01), Seattle, USA, September 10 - 13, 2001

[Chri2k] Christiansen, N.: IS THERE ANYBODY OUT THERE: Context awareness in a virtual organization. CHI2000 Workshop #11, The Hague, Netherlands, 01-06 April, 2000

[Coen91] Coen, M. A.: Design Principles for Intelligent Environments, Proceedings of AAAI 1998 Spring

Symposium on Intelligent Environments, Palo Alto, CA (March 23-25, 1998), pp. 36-43.

[Dec97] Digital Equipment Corporation: Universal Messaging Whitepaper. May 1997

[Dey2ka] Dey, A; Abowd, G.: Towards a Better Understanding of Context and Context-Awareness. CHI2000 Workshop #11, The Hague, Netherlands, 01-06 April, 2000

[Eck96] Eckardt, T.; Magedanz, T.; Ulbricht, C.; Popescu-Zeletin, R.: Generic Personal Communications Support for Open Service Environments. Proc. IFIP World Conference on Mobile Communications, Canberra, Australia, Sep. 1996

[Emm97] Emmerson, B.: Your Only Inbox. Byte Magazine, Dezember, 1997

[FAIN] A Policy –Based Management Approaches for Active and Programmable Networks http://www.ist-fain.org/deliverables/FAIN.paper.Vf.pdf

[Gins95] Ginsberg, A.; Ahuja, S.: Automating Envisionment of Virtual Meeting Room Histories, Proc. of ACM Multimedia 95 San Francisco, CA (November 5-9, 1995).

[Gre97] Greenberg, I.: The Universal Inbox. Byte Magazine, November, 1997

[Grosz89] Grosz, B.; Pollack, M.; Sidner, C.: Discourse, Foundations of Cognitive Science, M. Posner, Editor, MIT Press, Cambridge, MA (1989), pp. 65-75.

[Grub] Gruber, T. R.: Toward principles for the design of ontologies used for knowledge sharing., International Workshop on Formal Ontology, Padova, Italy. Available as technical report KSL-93-04, Knowledge Systems Laboratory, Stanford University.

[Har94] Harter, A.; Hopper, A.: A Distributed Location System for the Active Office. IEEE Network, Special Issue on Distributed Applications for Telecommunications, January 1994

[Hei] van Heijst, G.: The Role of Ontologies in Knowledge Engineering.

[Hov97] Hovorka, D.: Situation Analysis for the Development of Global Internet Access and Unified Messaging. eGlobe Ltd., Denver, USA, 1997

[IN] ITU-T Recommendations Q.1200 series: Intelligent Network. - Geneva, March 1992

[ISTAG01] IST Programme Advisory Group: SCENARIOS FOR AMBIENT INTELLIGENCE IN 2010, Final Report. IPTS-Seville, February 2001 (see www.cordis.lu/ist/istag.htm)

[Lamm2k] Lamming, M.; Eldridge, M.: Context-Based Systems: Research Challenges. CHI2000 Workshop #11, The Hague, Netherlands, 01-06 April, 2000

[Ljun2k] Ljungstrand, P.: Context-awareness in distributed communication systems, CHI Workshop #11: The Who, What, Where, When, Why and How of Context-Awareness, Tha Hague, April 2000

[Mag96] Magedanz, T.; Zeletin, R: Intelligent Networks. Basic Technology, Standards and Evolution. Int. Thomson Computer Press, London, 1996

[Martin99] Martin, D.: The Open Agent Architecture: A Framework for Building Distributed Software Systems, Applied Artificial Intelligence: An International Journal 13 No. 1-2, January-March 1999

[MDA] Siegel, J.; OMG Staff Strategy Group: Developing in OMG's Model-Driven Architecture. White Paper, Revision 2.6, Object Management Group, OMG document omg/01-12-01, November, 2001

[Mohr02] Mohr, W.: The Wireless World Research Forum (WWRF) Towards Systems Beyond 3G. The Proceedings of the Korean Institute of Communication Sciences, vol. 19, No. 7, July 2002, pp. 56, invited paper.

[Mohr03] Mohr, W.: The Wireless World Research Forum - WWRF. Computer Communications, Vol. 26, No. 1, January 2003, pp. 2 - 10, invited special issues.

[MS] Müller, H.J.; Schappert, A.: The Knowledge Factory - A Generic Knowledge Management Architecture

[Nagao94] Nagao, K.; Takeuchi, A.: Social Interaction: Multimodal Conversation with Social Agents, Proceedings of the 12th National Conference on Artificial Intelligence (AAAI-94), Seattle, WA (August 1-4, 1994), Vol. 1, pp. 22-28.

[Nan97] Nanneman, D.: Unified Messaging: A Progress Report. Telecommunications Magazine, March 1997

[Nor93] D. Norman, Things That Make Us Smart, Perseus Books, Reading, MA (1993).

[OMG-IDL] OMG Interface Definition Language Syntax and Semantic, Version 3.0, OMG document formal/02-06-39, available online at http://www.omg.org/cgi-bin/doc?formal/02-06-39

[ODP] Raymond, K.: Reference Model of Open Distributed Processing (RM-ODP): Introduction. - Center for Information Technology Research, University of Queensland, Australia; also in: Proc. of the International Conference on Open Distributed Processing, ICODP'95, Brisbane, Australia, pp. 20 - 24 February, 1995

[Pana2k] Panayiotou, C. Context Awareness (position paper). CHI2000 Werkstatt #11, The Hague, Netherlands, 01-06 April, 2000

[Pen99] Pentland, A.: Perceptual Intelligence. First International Symposium on Handheld and Ubiquitous Computing, HUC'99, Karlsruhe, Germany, 27-29 Sept., 1999

[Pent96] Pentland, A.: Smart Rooms, Scientific American 274 No. 4, 68-76 (April 1996).

[Pent99] Pentland, A.; Liu, A.: Modeling and Prediction of Human Behavior, Neural Computation 11 229-242 (1999).

[Per] Perez, A. G.: Overview of Knowledge Sharing and Reuse Components:Ontologies and Problem-Solving Methods, Proceedings of the IJCAI-99 workshop on Ontologies and Problem-Solving Methods (KRR5), Stockholm, Sweden, August 2, 1999

[Petr2k] Petrelli, D.; Not, E.; Strapparava, C.; Stock, O.; Zancanaro, M.: Modeling Context Is Like Taking Pictures, CHI2000 Workshop #11, The Hague, Netherlands, 01-06 April, 2000

[Pfe03] Pfeifer, T.; Popescu-Zeletin, R.: Seamless Integration of Distributed Internet Devices for Pervasive Architectures. Proc. of the 9th International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2003), IEEE Computer Society, San Juan, Puerto Rico, May 28-30, 2003, pp. 70 – 76, ISBN 0-7695-1910-5

[Pfe98] Pfeifer, T., v.d. Meer, S.: The Active Store providing Quality Enhanced Unified Messaging. 5th Conference on Computer Communications, AFRICOM-CCDC'98, Tunis, October 20-22-1998

[Pfe99] Pfeifer, T.: Automatic Conversion of Communication Media. - Dissertation, Technical University of Berlin, Institute for Open Communications Systems (OKS), 1999

[Rako2k] Rakotonirainy, A.; Loke, S.; Fitzpatrick, G.: Context-Awareness for the Mobile Environment. CHI2000 Workshop #11 Proposal, The Hague, Netherlands, 01-06 April, 2000

[SBF] Studer, R.; Benjamins, V.R.; Fensel, D.: Knowledge Engineering – Principles and Methods.

[SDO1] Fraunhofer FOKUS: Super Distributed Objects (SDO) – Specification. Deliverable 1, September 2001.

[SDO2] Fraunhofer FOKUS: Super Distributed Objects (SDO) – Specification. Deliverable 3, November 2001.

[SDO3] Fraunhofer FOKUS: Super Distributed Objects (SDO) – Specification of Components. Deliverable 4, April 2002.

[SDO4] Fraunhofer FOKUS: Super Distributed Objects (SDO) II D1.2 – Final use cases and requirements, November 2002

[SDO5] Fraunhofer FOKUS: Super Distributed Objects (SDO) II D3.1 – Specification of SDO functions, April 2003

[SDO-RFP] Object Management Group: RFP PIM and PSM for Super Distributed Objects. sdo/02-01-04

[SDO-Spec] Object Management Group: SDO for PIM and PSM draft adopted specification. dtc/03-04-02

[SDO-Sub] Fraunhofer FOKUS: Platform Independent Model (PIM) and Platform Specific Model (PSM) for Super Distributed Objects. Initial Submission to SDO RFP. sdo/02-09-01

[SDO-WP] Object Management Group: *Super Distributed Objects*. White paper. sdo/01-07-05

[Stif96] Stifelman, L.: Augmenting Real-World Objects: A Paper-Based Audio Notebook, Proc. of CHI '96 Vancouver, BC (April 13-18, 1996).

[Tsib2k] Tsibidis, G.; Arvanitis, T.; Baber, C.: The What, Who, Where, When, Why and How of Context-Awareness. CHI2000 Workshop #11, The Hague, Netherlands, 01-06 April, 2000

[UMTS] ETSI Techn. specification TS 22.01v.3.1.0: Universal Mobile Telecommunication Systems (UMTS). Aspects, Principles. - Sophia Antipolis, France, 1997

[VHE] ETSI Draft 22.70v.0.0.3: Virtual Home Environments. - Sophia Antapolis, France, 1997

[War97] Ward, A.; Jones, A.; Hopper, A.: A New Location Technique for the Active Office. IEEE Personal Communications; Vol. 4 (1997) 5; New York: IEEE, Oct. 1997, pp. 42-47

[Water00] Waterstrat, H.: Managing Context. - Diplomarbeit, Technical University of Berlin, Institute for Open Communications Systems (OKS), 2000

[Wechs99] Wechselberger, R.: The Application of Natural Language Dialogue Processing for the Remote Access to a Unified Messaging System. - Diplomarbeit, Technical University of Berlin, Institute for Open Communications Systems (OKS), 1999

[Weiser91] Weiser, M.: The Computer for the 21st Century, Scientific American 265 No. 3, 94-104 (September 1991).

[Well93a] Wellner, P.; Mackay, W.; Gold, R.: Computer Augmented Environments: Back to the Real World, Communications of the ACM 36 No. 7, 24-26 (July 1993).

[Well93b] Wellner, P.: Interacting with Paper on the Digital Desk, Communications of the ACM 36 No. 7, 87-96 (July 1993).

[Win98] Winokur, D. M.: The Internet as a Global Messaging Platform. Spring Internet World, 1998

[X.901] ITU-T Recommendation X.901: Information technology - Open Distributed Processing - Reference Model: Overview. - Geneva, Aug. 1997, (ISO/IEC IS 10746-1)

[X.902] ITU-T Recommendation X.902: Information technology - Open Distributed Processing - Reference Model: Foundations. - Geneva, Nov. 1995, (ISO/IEC IS 10746-2)

[X.903] ITU-T Recommendation X.903: Information technology - Open Distributed Processing - Reference Model: Architecture. - Geneva, Nov. 1995, (ISO/IEC IS 10746-3)

[X.904] ITU-T Recommendation X.904: Information technology - Open Distributed Processing - Reference Model: Architectural Semantics. - Geneva, 1998, (ISO/IEC IS 10746-4)

[Z.130] ITU-T Recommendation X.130: Extended object definition language - Techniques for Distributed Software Component Development Conceptual Foundation, Notation, and Technology Mappings. – Geneva, February 2003

[Zel98] Popescu-Zeletin, R.; Pfeifer, T.; Magedanz, T.: Applying Location-Aware Computing for Electronic Commerce: Mobile Guide. Proc. of the 5th Conference on Computer Communications, AFRICOM-CCDC'98, Tunis, October 20-22-1998

6.2 WWW-Links

[W-3GPP] 3rd Generation Partnership Project
http://www.3gpp.org/

[W-ACL] Agent Communication Language
http://www.fipa.org/specs/fipa00061/

[W-AOS] FIPA Ontology Service Specification
http://www.fipa.org/specs/fipa00086/

[W-BoV01] WWRF: Book of Visions, Edition December 2001
http://www.wireless-world-research.org/

[W-CORBA] Common Object Request Broker Architecture
http://www.omg.org/technology/documents/formal/corba_iiop.htm

[W-DubC] Dublin Core Metadata Initiative
http://dublincore.org/

[W-Echonet] Echonet Consortium
http://www.echonet.gr.jp/english/

[W-FIPA] Foundation for Intelligent physical Agents
http://www.fipa.org

[W-FSAN] Full Service Access Networks Initiative
http://fsan.mblast.com/initiative.asp

[W-HAVi] Home Audio Video Interoperability
http://www.havi.org/

[W-Jini] Jini
http://www.jini.org/

[W-JXTA] Project JXTA
http://www.jxta.org

[W-KBS] Knowledge Based Systems
http://www.brighton.ac.uk/kes/journal/

[W-Lang] Survey of the State of the Art in Human Language Technology
http://www.cslu.ogi.edu/HLTsurvey/

[W-OCL] OMG Object Constraint Language Specification
http://www.omg.org/cgi-bin/doc?formal/03-03-13

[W-OMA] Open Mobile Alliance
http://www.openmobilealliance.org/

[W-OMG-NS] OMG CORBAservices - Naming Service
http://www.omg.org/cgi-bin/apps/do_doc?formal/97-12-10.pdf

[W-OPS] Open Profiling Standard
http://www.w3.org/TR/NOTE-OPS-FrameWork.html
http://developer.netscape.com/ops/ops.html

[W-OSA] OSA/Parlay Group
http://www.parlay.org/

[W-OSGi] The Open Services Gateway Initiative
http://www.osgi.org/

[W-OSI] Open System Interconnection Model, ISO 7498
http://www.acm.org/sigcomm/standards/iso_stds/OSI_MODEL/

[W-OSP] KSL Ontology Server Projects
ontolingua.stanford.edu

[W-P3P] Platform for Privacy Preferences Project
http://www.w3.org/P3P/

[W-P3P] Privacy and Profiling on the Web
http://www.w3.org/TR/NOTE-Web-privacy.html

[W-Parlay] OSA/Parlay Group
http://www.parlay.org/

[W-Perv1] IBM Research on Pervasive Computing
http://www.research.ibm.com/compsci/communications/projects/pervasive/index.htm

[W-Perv2] Turning pervasive computing into mediated spaces
http://www.research.ibm.com/journal/sj/384/mark.html

[W-RDF] W3C Resource Description Framework
http://www.w3.org/RDF/

[W-SemW] Semantic Web Community Portal
http://www.semanticweb.org/

[W-SmartD] MIT Media Lab: Vision and Modeling Group's Smart Desk Project
vismod.www.media.mit.edu/vismod/demos/smartdesk

[W-TINA-BM] Tina-Consortium: TINA Business Model and Reference Points
http://www.tinac.com/specifications/documents/bm_rp.pdf

[W-TINA-C] Tina-Consortium
http://www.tinac.com

[W-TINA-SA] TINA-C: Baseline Service Architecture
http://www.tinac.com/specifications/documents/sa50-main.pdf

[W-TTT] MIT: Things That Think
http://ttt.media.mit.edu/

[W-UbiCom] Ubiquitous Computing
http://www.ubiq.com/hypertext/weiser/UbiHome.html

[W-UMS] Jfax: http://www.jfax.com
3Box: http://www.3box.de
Tobit: http://www.tobit.com
Web.de: http://www.web.de

[W-UNL] Universal Networking Language
http://www.unl.ias.unu.edu/publications/UNL-beyond%20MT.html

[W-UPnP] UPnP™ Forum
http://www.upnp.org/

[W-WG2-AA] WWRF WG2: White Paper on Ambient-awareness
http://www.comtec.e-technik.uni-kassel.de/content/conference/wwrf-wg2/

[W-WG2-Ad] WWRF WG2: White Paper on Service Adaptability
http://www.comtec.e-technik.uni-kassel.de/content/conference/wwrf-wg2/

[W-WG2-BM] WWRF WG2: White Paper on Business Models
http://www.comtec.e-technik.uni-kassel.de/content/conference/wwrf-wg2/

[W-WG2-Per] WWRF WG2: White Paper on Service Personalization
http://www.comtec.e-technik.uni-kassel.de/content/conference/wwrf-wg2/

[W-WSDL] W3C Web Services Description Language
http://www.w3.org/TR/wsdl

[W-WSI] IST Wireless Strategic Initiative
http://www.ist-wsi.org/

[W-WWRF] Wireless World Research Forum
http://www.wireless-world-research.org/

[W-WWRI] Wireless World Research Initiative
http://www.ist-wwri.org/

[W-XSLT] W3C XSL Transformations
http://www.w3.org/TR/xslt

[W-ZigBee] ZigBee™ Alliance
http://www.zigbee.org/

6.3 Acronyms

3Gb	3rd Generation beyond
3GPP	3rd Generation Partnership Project
ACL	Agent Communication Language
all-IP	All Internet Protocol
API	Application Programming Interface
ASR	Automatic Speech Recognition
BAN	Body Area Network
BMBF	Bundesministerium für Bildung und Forschung
CE	Communication Elements
CORBA	Common Object Request Broker Architecture
CS	Context Server
DHCP	Dynamic Host Configuration Protocol
DNS	Domain Name System
DPE	Distributed Processing Environment
DSIG	Domain Special Interest Group
E-Book	Electronic Book
Echonet	no acronym – Japanese standard for device interconnectivity
ETSI	European Telecommunication Standardization Institute
FIPA	Foundation for Intelligent Physical Agents
FOKUS	Forschungsinstitut für Offene Kommunikationssysteme
GPRS	General Packet Radio System
GPS	Global Positioning System
GSE	Generic Service Element
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HAVi	Home Audio Video Interoperability (www.havi.org)
HMI	Human Machine Interface
HSCSD	High Speed Circuit Switched Data
HTTP	Hypertext Transfer Protocol
I/O	Input / Output
ID	Identifier
IDL	Interface Definition Language
IMT-2000	International Mobile Telecommunication 2000
I-net	Individual-Centric Network
IP	Internet Protocol
IPv4	Internet Protocol version 4
IPv6	Internet Protocol version 6
IrDA	Infrared Device Access
ISDN	Integrated Service Digital Network
IST	Information Society Technologies
ISTAG	IST Programme Advisory Group
ITU	International Telecommunication Union
IVR	Interactive Voice Response
J2SE	Java ™ 2 Standard Edition
JDK	Java Development Kit
Jini	no acronym – international standard for device interconnectivity
JXTA	Is not an acronym; the word is derived from the word juxtapose
KBS	Knowledge based system
LAN	Local Area Network
MDA	Model Driven Architecture
MSC	Message Sequence Chart
ODBC	Open Database Connectivity
ODL	Object Definition Language
ODP	Open Distributed Processing
OMA	Open Mobile Alliance
OMG	Object Management Group
OPF	Open Profiling Framework
OSA	Open Service Access
OSGi	Open Service Gateway Initiative
OSI	Open System Interconnection
P2P	Peer-to-Peer
PAN	Personal Area Network
PDA	Personal Digital Assistant
PIM	Platform Independent Model
PSM	Platform Specific Model
PSTN	Public Switched Telephony Networks
QoS	Quality of Service
RDF	Resource Description Framework
RF	Radio Frequency
RGB	red green blue
SDK	Software Development Kit
SDO	Super Distributed Object
SDP	Service Discovery Protocol
SIG	Special Interest Group
SLA	Service Level Agreement
SMS	Short Message Service
SQL	Structured Query Language
TINA	Telecommunication Information Networking Architecture
TUB	Technical University Berlin
TTS	Text-to-Speech
TV	Television
UI	User Interface
UML	Unified Modeling Language
UMS	Unified Messaging System
UMTS	Universal Mobile Telecommunication System
UPnP	Universal Plug and Play
UUID	Universally Unique Identifier
VHE	Virtual Home Environment

W3C	World Wide Web Consortium
WAN	Wide Area Network
WLAN	Wireless Local Area Network
WSDL	Web Service Description Language
WSI	Wireless Strategic Initiative
WWRF	Wireless World Research Forum
WWRI	Wireless World Research Initiative
XML	Extensible Markup Language
XPATH	XML Path
XSD	XML Style Sheet Document
XSL	XML Style Sheet Language
XSLT	XML Style Sheet Language Transformation

7 Appendix

7.1 Enabling Technologies

The following list of technologies represents a selection of technologies, which mostly influenced the design and implementation of the Open Profiling Framework.

7.1.1 Ontology Description Languages

The next section gives examples of description languages. It can be distinguished between languages, which were made to describe knowledge and only that, e.g. Ontology Inference Layer (OIL), Knowledge Interchange Format (KIF), or Conceptual Graph (CG). These languages support all needed concepts, like classes, relation, functions, axioms, and instances, to describe knowledge. Other languages can also used to describe knowledge respectively information, e.g. RDF/RDFS or XML/XML Schema. RDF/RDFS support only a part of needed concepts, like classes, relations, and instances. XML/XML Schema support indirect classes (XML Schema) and instances (XML document). XML (the standard itself) supports none of them. Generally, RDF/RDFS and XML/XML Schema can extended by additive definitions to compensate lacking concepts. Summarizing it can be asserted that it does not give 'the language' to describe knowledge or information. For each case is to decide, which language is the right.

XML, and its grammar formalisms describe SYNTAX (the physical ordering of elements), but not SEMANTICS (meaning of elements). RDF is a language for creating meta-data. OIL was designed specifically for building ontologies that have more expressiveness than RDF.

7.1.2 WebOnt

The W3C WebOnt working group, part of the Semantic Web Activity, focuses on the development of a language to extend the semantic reach of current XML and RDF meta-data efforts. The ontological layer and the formal underpinnings thereof will be addressed, which is needed for developing applications that depend on an understanding of logical content, not just human-readable presentation. Ontology in the context of this work refers to what is sometimes called a 'structural' ontology; a machine-readable set of definitions that create taxonomy of classes and subclasses and relationships between them.

Such language layers are crucial to the emerging Semantic Web, as they allow the explicit representation of term vocabularies and the relationships between entities in these vocabularies. In this way, they go beyond XML, RDF and RDF-S in allowing greater machine readable content on the web. A further necessity is for such languages to be based on a clear semantics (denotation and/or axiomatic) to allow tool developers and language designers to unambiguously specify the expected meaning of the semantic content when rendered in the Web Ontology syntax.

7.1.3 Semantic Web

The World Wide Web is a collection of many HTML-sites, which are contain enormous knowledge. However, this knowledge is not available in an easy manner, because the knowledge is not structured. The reading of all sites offers the whole knowledge. A direct access of the knowledge is not possible. Search engines are a first approach to give a better access of the Web-knowledge. So-called Web-robots 'read' all sites and subscript the HTML-site by defined categories. This subscription is very vaguely, because the HTML-sites contain 'plain'-text. That makes it very difficult to understanding the text-content for the robots. A Web Site-Ontology, which offers a common vocabulary for site description, could help understanding of Web sites. That makes it possible to build better indices and therefore to offer a 'better' (more exact) access of Web-knowledge.

7.1.4 Metadata for Electronic Resources

The Dublin Core [W-DubC] is a metadata element set intended to facilitate discovery of electronic resources. Originally conceived for author-generated description of Web resources, it has attracted the attention of formal resource description communities such as museums, libraries, government agencies, and commercial organizations.

The Dublin Core Workshop Series has gathered experts from the library world, the networking and digital library research communities, and a variety of content specialties in a series of invitational workshops. The building of an interdisciplinary, international consensus around a core element set is the central feature of the Dublin Core. The progress represents the emergent wisdom and collective experience of many stakeholders in the resource description arena. An open mailing list supports ongoing work.

The characteristics of the Dublin Core that distinguish it as a prominent candidate for description of electronic resources fall into several categories:

- Simplicity: The Dublin Core is intended to be usable by non-catalogers as well as resource description specialists. Most of the elements have a commonly understood semantics of roughly the complexity of a library catalog card.
- Semantic Interoperability: In the Internet Commons, disparate description models interfere with the ability to search across discipline boundaries. Promoting a commonly understood set of descriptors that helps to unify other data content standards increases the possibility of semantic interoperability across disciplines.
- International Consensus: Recognition of the international scope of resource discovery on the Web is critical to the development of effective discovery infrastructure. The Dublin Core benefits from active participation and promotion in some 20 countries in North America, Europe, Australia, and Asia.
- Extensibility: The Dublin Core provides an economical alternative to elaborate more description models such as the full MARC cataloging of the library world. Additionally, it includes sufficient flexibility and extensibility to encode the structure and more elaborate semantics inherent in richer description standards
- Metadata Modularity on the Web: The diversity of metadata needs on the Web requires an infrastructure that supports the coexistence of complementary, independently maintained metadata packages. The World Wide Web Consortium (W3C) has begun implementing an architecture for metadata for the Web. The Resource Description Framework, or RDF, is designed to support the many different metadata needs of vendors and information providers. Representatives of the Dublin Core effort are actively involved in the development of this architecture, bringing the digital library perspective to bear on this important component of the Web infrastructure.

7.1.5 Open Profiling Standard

The Open Profiling Standard (OPS) proposes a means for the exchange of profile information. It supports the creation of contexts for personalization with privacy management. For the purposes of OPS, profile information is defined as any feature and corresponding values of an end user or service provider. The specification provides for the trusted communication:

- between people and services
- between services mediated by people, and
- between people.

The ‘Proposal for an Open Profiling Standard’, covers the framework for profile exchange, including data structure and operational primitives. The ‘Standard OPS Practices’ covers ‘well-known sections’ (standard profile elements), suggested permission mechanisms for user and service provider control of the exchange of profile information, and standard transaction logging techniques. [W-OPS]

7.1.6 XML – eXtensible Markup Language

An interesting trend in describing data structures is the application of semi-structured description languages like the XML. They are capable to describe nearly each kind of data structure in a human readable way. Beside the semi-structured description, a Document Type Definition (DTD) is needed, which defines how data structures have to be described. In a context-aware system, a DTD can define an alphabet and a grammar for the description of specific contexts. All profiles within these contexts can be described using the same DTD. If a profile is to be extended, only the corresponding DTD has to be changed. [XML]

The *eXtensible Markup Language* (XML) is a restricted form of the *Standard Generalized Markup Language* (SGML) and has been developed by an XML Working Group formed under the auspices of the World Wide Web Consortium (W3C) in 1996.

The XML standard provides a set of rules for storing structured data in a way that produces text documents that are easy to generate and read (by a computer system as well as by a human[37]). Because they are just formatted text files, XML documents are flexible, easy to extend, and platform independent. XML documents consist of character data, which forms the content, and markup data, which encodes the document's logical structure. XML uses so called *tags* (words bracketed by '<' and '>') to distinguish the content data from additional information. The use of tags is shown in the following example.

```
<Heading chapter="7"> Technology viewpoint </Heading>
```

The XML data inside the sample tag above describes the headline of this chapter. Additional information about this character data is provided with the tag `Heading` and the attribute `chapter`. The usage of tags is very common in the Internet community since they are also used to add layout information to Hypertext documents written in the *Hypertext Markup Language* (HTML). This language specifies what each tag and attribute mean in the context of the document's layout and in which order they can appear. Therefore, tags in HTML express a certain well-defined meaning. In opposite to that, XML uses tags only to delimit pieces of data, and leaves the interpretation of the data completely to the application that consumes it.

The difference between HTML and XML results from the purposes of each language. HTML is intended to specify a document, which makes it necessary to define a fixed set of tags that have a certain meaning in terms of formatting, structuring, and layout. The only purpose of tags in XML is to distinguish content from structuring information. Therefore the tag `Heading` and its attribute `chapter` can mean that the information inside it forms the headline of the third chapter. However, leaving the interpretation open to the applied application, it can mean something very different as well.

```
<?xml version="1.0"?>
<REFERENCE_LIST>
   <BOOK>
  <AUTHOR> Harold, Elliotte Rusty </AUTHOR>
  <TITLE> XML Bible </TITLE>
  <ISBN> 0764532367 </ISBN>
  <PUBLICATION year="1999"/>
    </BOOK>
    <BOOK>
  <AUTHOR> Kay, Michael </AUTHOR>
  <TITLE> XSLT Programmer's Reference </TITLE>
  <ISBN> 1861003129 </ISBN>
  <PUBLICATION year="2000"/>
    </BOOK>
</REFERENCE_LIST>
```

[37] Human-readable in this context does not mean that everybody can easily understand the content of an automatically generated XML file. In most cases, such XML files are quite long and contain confusing information. It rather means that a person who is familiar with XML in general is able to debug or analyze an unknown XML file by opening and editing it with a simple text editor.

The more complex example above demonstrates the structuring capabilities of XML. It corresponds to the reference list of this work and comprises two books: The *XML Bible* and the *XSLT Programmer's Reference*. Information about these books is divided into four parts: the author (`AUTHOR`), the book's title (`TITLE`), its ISBN (`ISBN`), and year of publication (`PUBLICATION`). A tag with the corresponding name represents each of these parts. Note that there are different ways of expressing character data: Inside a tag (like the name of the author) and as an attribute (like the year of publication).

Such an XML document can be easily exchanged between different applications. In the context of programming languages, XML data is always a string. Every application can process and interpret this data, omitting irrelevant information and format useful information in its own style. For instance, if an HTML representation of this reference list is needed, it can be transformed to an HTML list that contains author and title of all books.

XML also provides a mechanism to impose constraints on the storage layout and logical structure using *XML Schema*. Xml schema not only force elements and attributes relation within the document, but also enforce specific data type restrictions. It supports user-defined complex data types, data ranges, and masks. Xml schema of the *reference-list*, the example shown above, the *reference-list* described above can look like as the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
  <xs:element name="REFERENCE_LIST">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="BOOK" maxOccurs="unbounded">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="AUTHOR" type="xs:string"/>
              <xs:element name="TITLE" type="xs:string"/>
              <xs:element name="ISBN" type="xs:integer"/>
              <xs:element name="PUBLICATION">
                <xs:complexType>
                  <xs:attribute name="year" type="xs:gYear" use="required"/>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

It can be seen here that the structure of the XML Schema is the same as that of any other XML document. The schema defines that the document conform to this schema must have the root element with the name '*Preference-List*'. It enforces that this root element to have one of more element *BOOK*. A *BOOK* element has four child elements. The first tree element is of type *String*. The fourth element has the attribute with the name *year* is of type *year*, defined in the XML Schema definition language.

An XML document that fulfills the rules of the XML specification is called *well formed*. A well-formed XML document that conforms to the specified *schema* is called *valid*. Therefore, the reference list example is valid in respect to the above schema. [38]

XML serves as a basis for the open profiling concept as described in this document. Primarily the extensibility of XML is of great use, because it enables to cope with the variability of profiles. In an I-centric communications system, a *schema* can define the structure and the content for the description of specific contexts. All profiles within these contexts consist of XML data and can be described using the same *schema*. If the abstract model behind a profile changes,

[38] Actually the XML example misses a reference to the imposed *schema* to be fully valid. A valid XML document must have a reference to the corresponding *schema* document.

only the corresponding *schema* has to be adapted. New XML data that reflects the *schema* automatically conforms to the changed model.

7.1.7 RDF – Resource Description Framework

The Resource Description Framework (RDF) [W-RDF] is a proposed standard for processing metadata, which has been developed under the auspices of the World Wide Web Consortium. It provides an infrastructure that enables the encoding, exchange, and reuse of structured metadata and therefore helps to increase interoperability between applications that exchange machine-understandable information.

RDF can be used in a variety of application areas, for example:

- to catalogue or describe the content and content-relationships available at a particular Web site, document, or digital library;
- to facilitate knowledge sharing and exchange for intelligent software agents;
- to describe collections of text parts that represent a single logical document;
- to express the privacy preferences of a user.

RDF uses XML as a common syntax for the exchange and processing of metadata about any type of resource (including XML and non-XML resources). The RDF data model defines a simple model for describing interrelationships among resources in terms of named properties and values. The model consists of three object types: resources, properties, and statements.

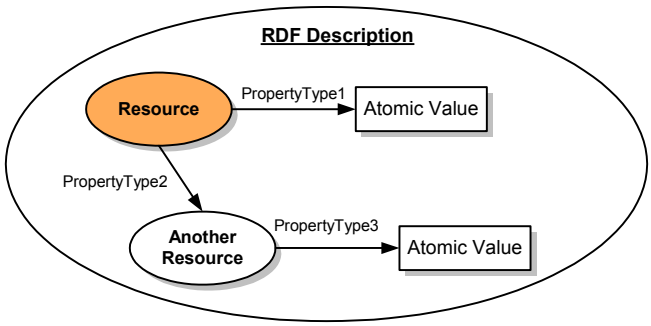


Figure 135: Example for RDF Description Model

All items being described by RDF expressions are called **resources**. A resource may be an entire Web page (e.g. the HTML document http://www.w3.org/Overview.html), a part of a Web page (e.g. a specific HTML or XML element within the document source), a complete collection of pages (e.g. an entire Web site), or an object that is not directly accessible through the Web (e.g. a printed book).

A **property** is a specific aspect, characteristic, attribute, or relation used to describe a resource. Each property has a specific meaning and defines its permitted values as well as its relationship with other properties. In RDF, values can be atomic (text strings, numbers, etc.) or other resources, which may have their own properties, too.

An RDF **statement** contains a specific resource together with a property name and a property value. These three individual parts of a statement are called the subject, the predicate, and the object. The object of a statement (i.e. the property value) can be another resource, a literal, a simple text string, or a primitive data type defined by XML.

In general, more than one statement describes a given resource. For instance, if a book is described, information like title, author, and ISBN are important. For this superficial description already, three statements are needed. Therefore, RDF offers the possibility to group such statements together in a **description**.

An RDF Description is a placeholder for one or more RDF Statements concerning a certain resource. It provides a way to specify the subject (e.g. the resource) just once for several statements and group them together. Therefore, a Description forms a collection of properties that

refer to the same resource. Figure 135 gives an example for an RDF Description that holds several statements about the Resource. The Description comprises three Statements: two for the actual described Resource and one for AnotherResource.

The example below shows the reference list example from the previous section as an RDF Description. Since RDF uses XML as a common syntax, the RDF description looks partially similar to the XML example. Nevertheless, a deeper look unveils the differences between both examples. A main difference refers to the use of XML namespaces, which are attached to the RDF tag as attributes (`xmlns:rdf` and `xmlns:s`). These namespaces provide a reference to a *schema* resource, which describes the syntax of the elements like an XML DTD. All tags starting with '`rdf:`' represent RDF entities, while all tags starting with '`s:`' represent entities of a hypothetical reference list description schema.

```
<?xml version="1.0"?>
<rdf:RDF   xmlns:rdf="http://www.w3.org/RDF/RDF"
           xmlns:s="http://description.org/schema" >
  <rdf:Description ID="#reference_list">                (1)
    <s:BOOK REFERENCES>
      <rdf:Bag>
        <rdf:li resource="#Har99"/>
        <rdf:li resource="#Kay00"/>
      </rdf:bag>
    </s:BOOK REFERENCES>
  </rdf:Description>
  <rdf:Description ID="#Har99">                         (2)
    <s:TITLE> The XML Bible </s:TITLE>
    <s:AUTHOR> Harold, Elliotte Rusty </s:AUTHOR>
  </rdf:Description>
  <rdf:Description ID="#Kay00">                         (3)
    <s:TITLE> XSLT Programmer's Reference </s:TITLE>
    <s:AUTHOR> Kay, Michael </s:AUTHOR>
  </rdf:Description>
</rdf:RDF>
```

The RDF example shows three different resources that are described: the reference list (1) and the two books from the XML example (2, 3). The resources itself are identified with a Uniform Resource Identifier (URI), which are represented by the text strings '`#reference_list`', '`#Har99`', and '`#Kay00`'. The two book resources have two properties each: a `TITLE` and an `AUTHOR`. The values of these properties are atomic (text strings). The reference list resource has only one property: `BOOK_REFERENCES`. The value of this property contains an unordered list of resources, an RDF bag. This bag contains two references to book resources, which are identified by URI strings.

The RDF data model can provide an open and generic basis for the description of contextual information inside profiles. Since it is based on XML, it offers the same flexibility and extensibility.

7.1.8 XSLT – eXtensible Stylesheet Transformation

XSLT, which stands for *eXtensible Stylesheet Language Transformations*, is a proposed standard developed under the auspices of the World Wide Web Consortium. According to the specification, '*XSLT* […] *is a language for transforming XML documents into other XML documents*'. It allows programmers to selectively query, display, and manipulate data. It is also possible to perform scripting-like operations on the XML document and transform it into HTML, XML, or any other text-based format.

The *eXtensible Stylesheet Language* (XSL) includes both a transformation language and a formatting language. The transformation language provides elements that define rules for transforming one XML document into another, while the formatting language specifies an XML vocabulary for formatting semantics. Each of these languages is an XML application, i.e. the instructions for both languages are compliant to the XML standard. A transformation applying the *eXtensible Stylesheet Language* is therefore expressed as a well-formed XML document.

The transformation language is useful independently from the formatting language. Its ability to move data from one XML representation to another makes it an important component of XML

based electronic commerce, electronic data interchange, metadata exchange, and any application that needs to convert between different XML representations. Additionally, these use cases have a lack of concern with rendering data on a display for humans in common. Open profiling is neither concerned with formatting of nor with displaying data, so only the transformation part of XSL is described in more detail.

A transformation expressed in XSLT describes rules for transforming an XML source tree into a result tree. The transformation is achieved by associating certain patterns with templates. The patterns are matched against elements in the source tree, while templates are instantiated to create parts of the result tree. The structure of the result tree can be completely different from the structure of the source tree. In constructing the result tree, elements from the source tree can be filtered and reordered, and arbitrary structures can be added.

The following example of an XSL Transformation uses the XML data from the reference list example above. It shows an XSL Transformation that only uses the transformation language to alter the structure of the XML reference list while filtering the data at the same time. The transformation generates an XML file with a compact list of all books in the reference list that have been published in the year 1999.

```
<?xml version="1.0"?>
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
  <xsl:template match="REFERENCE_LIST">                                (1)
    <LIST year="1999">
      <xsl:apply-templates select="BOOK"/>
    </LIST>
  </xsl:template>
  <xsl:template match="BOOK">                                          (2)
    <xsl:if test="PUBLICATION/@year = '1999'">
      <BOOK title="{TITLE/text()}" author="{AUTHOR/text()}"/>
    </xsl:if>
  </xsl:template>
</xsl:stylesheet>
```

The XML data of this example use XML namespaces like the RDF example above. Tags starting with the prefix ‘`xsl:`’ are transformation language instructions, while tags without any prefix are data tags that are just copied to the output XML tree. An important instruction in this example is `xsl:template`. It provides a structure matching mechanism that selects certain parts of the source XML file for processing. In this case, the template instruction (1) matches all `REFERENCE_LIST` tags in the source tree, while the other template instruction (2) matches all `BOOK` tags.

The XSL Transformation process parses the XML source file from the beginning to the ending, starting with the uppermost XML tag. Whenever a tag matches a template, it is executed. The example transformation will therefore find a match for the tag `REFERENCE_LIST` first. It then executes the instruction inside the template (1), which states the following: Since the `LIST` tag has no XSL prefix, it is copied to the output XML tree. The output of this transformation is shown below. The output XML file therefore starts with the beginning `LIST` tag.

```
<?xml version="1.0"?>
<LIST year="1999">
  <BOOK title="XML Bible" author="Harold, Elliotte Rusty"/>
</LIST>
```

The instruction `xsl:apply-template` forces a template matching process on all `BOOK` tags inside the current `REFERENCE_LIST` tag. Now the book template is executed for all of these tags in the source XML tree. The instruction `xsl:if` performs a test, whether the publication year is equal to 1999. In this case, a `BOOK` tag is copied to the output, which has two attributes that hold the author and the title of the book. Otherwise, it will do nothing. The instructions used for the if-decision and the selection of the author and title character data are formed through the use of the XML Path Language (XPath), which is described in detail in the next section.

After the processing of all books inside the reference list, execution of the template (1) is continued and the end tag for `LIST` is written to the output tree. When execution of this template has finished, the transformation process finishes also, because there is no more data after the `REFERENCE_LIST` end tag in the XML source tree. The resulting XML tree consists of a list of all books published in 1999 after the transformation process. Since there is only one such book in the reference list XML example, the output list has only one `BOOK` tag: the 'XML Bible' from E.R. Harold.

The XSL transformation language offers several instructions for value comparison, decision-making, loops, and variable usage. It is therefore possible to encode algorithms with this language, which can process given XML data. In the context of open profiling, XSL Transformation can be used for profile evaluation, because it can generically process XML data with algorithms that have been specified in the XSL transformation language. The algorithm can then be interpreted by an XSL interpreter. Since the algorithm is specified independently from any interpreter, it can be changed during runtime without the need to rebuild the component.

7.1.9 XPath – XML Path Language

The XML Path Language (XPath) is used by XSLT as a sub language for addressing parts of an XML document. The language has been developed under the auspices of the World Wide Web Consortium.

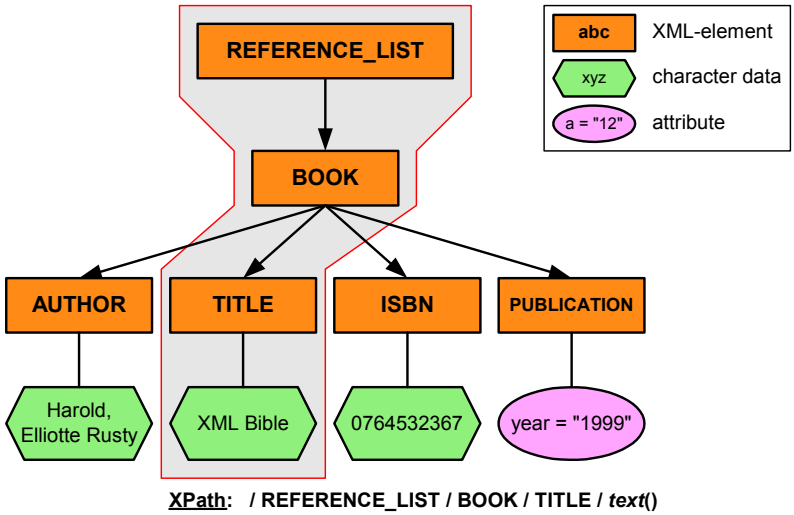


Figure 136: XPath Example: Reference List

The primary purpose of XPath is to provide a common syntax and semantic for the functionality of addressing parts of an XML document. XPath operates on the abstract, logical structure of an XML document, rather than on its surface syntax. XPath models an XML document as a tree of nodes, which can be element nodes, attribute nodes, or text nodes.

The XPath example shown in Figure 136 contains a tree representation of the first book in the reference list of the XML example introduced above. It illustrates, how the title of this book is identified with an XPath expression ('`/REFERENCE_LIST/BOOK/TITLE/text()`'). Such an expression denotes a path through the XML tree that points to a certain element, attribute, or text node. The individual element nodes in the XPath string are separated by a slash ('`/`') and identified over the element name. To access the character data (e.g. the text node) of an element, the XPath function '`text()`' has to be used. There are also several different functions in XPath that can be used to identify further certain parts of the tree.

An XPath expression may additionally be used for numerical calculations, string manipulations, or for testing Boolean conditions, but its most characteristic use (and the one that gives it its name) is to identify parts of the input XML document that is to be processed.

7.1.10 XML Schema

XML Schema is a definition language, which offers facilities for describing the structure and constraining the contents of XML 1.0 documents, including those, which exploit the XML Namespace facility. The schema language, which is represented in XML 1.0 and uses namespaces, substantially reconstructs and considerably extends the capabilities found in XML 1.0 document type definitions (DTDs). Any application that consumes well-formed XML can use the XML Schema formalism to express syntactic, structural and value constraints applicable to its document instances.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
  <xs:element name="Author" type="xs:string"/>
  <xs:element name="Book">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="Title"/>
        <xs:element ref="Author"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="Title" type="xs:string"/>
</xs:schema>
```

The XML Schema formalism allows a useful level of constraint checking to be described and implemented for a wide spectrum of XML applications. An XML Schema consists of components such as type definitions and element declarations. These can be used to assess the validity of well-formed element and attribute information items, and furthermore may specify augmentations to those items and their descendants. This augmentation makes explicit information, which may have been implicit in the original document, such as normalized, and/or default values for attributes and elements and the types of element and attribute information items.

```
<?xml version="1.0" encoding="UTF-8"?>
<Book xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xsi:noNamespaceSchemaLocation="books.xsd">
  <Title>Moby Dick</Title>
  <Author>Dickens</Author>
</Book>
```

The examples given here show a schema definition and an XML document, which is valid against this definition.

7.2 Open Profiling Framework – Schema Specifications

This annex contains the XML-Schema files describing the structure of the data used inside the Open Profiling Framework.

7.2.1 SDO parameters

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
 <xs:element name="EXECUTE">
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="PARAMETER" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="OPERATIONNAME" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="PARAMETER">
  <xs:complexType>
   <xs:attribute name="NAME" type="xs:string" use="required"/>
   <xs:attribute name="VALUE" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

7.2.2 Service Configuration

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:include schemaLocation="D:\SDO\dtd\GetData.xsd"/>
 <xs:include schemaLocation="D:\SDO\dtd\Subscribe.xsd"/>
 <xs:include schemaLocation="D:\SDO\dtd\Notify.xsd"/>
 <xs:element name="CONFIGURATION">
  <xs:annotation>
   <xs:documentation>Root element of the configuration xml file.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="NOTIFY" minOccurs="0"/>
    <xs:element ref="INITIALSTATICDATA"/>
    <xs:element ref="SUBSCRIPTIONS"/>
    <xs:element ref="BEHAVIOR"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="INITIALSTATICDATA">
  <xs:annotation>
   <xs:documentation>Desc. of the static data to get from CS while
initialising.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="GETDATA" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="SUBSCRIPTIONS">
  <xs:annotation>
   <xs:documentation>List of Subscribe elements to subscribe at the Context
Server.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="SUBSCRIBE" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="BEHAVIOR">
  <xs:annotation>
   <xs:documentation>Describes how the service should react upon getting subscribed
data from the Context Server.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="RULE" minOccurs="0" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="CONDITION">
  <xs:annotation>
   <xs:documentation>Context of the data as the Condition for the rule. If the data
received is of the specified context then this rule is used.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="CONTEXT"/>
   </xs:sequence>
   <xs:attribute name="InformationType" use="required">
    <xs:annotation>
     <xs:documentation> Can be either 'SDO' or 'Relation'.</xs:documentation>
```

```
    </xs:annotation>
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="ANY"/>
      <xs:enumeration value="SDODATA"/>
      <xs:enumeration value="RELATION"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="ChangeType" use="required">
    <xs:annotation>
     <xs:documentation>Can be 'Insert', 'modify', 'delete', or 'system'
data.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="ANY"/>
      <xs:enumeration value="INSERTDATA"/>
      <xs:enumeration value="MODIFYDATA"/>
      <xs:enumeration value="DELETEDATA"/>
      <xs:enumeration value="SYSTEMDATA"/>
      <xs:enumeration value="REGISTERDATA"/>
      <xs:enumeration value="DEREGISTERDATA"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:complexType>
 </xs:element>
 <xs:element name="ACTION">
  <xs:annotation>
   <xs:documentation>Action of the service upon getting specific
data.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="RELOADINITIALDATA" minOccurs="0"/>
    <xs:element ref="GETDATALIST" minOccurs="0"/>
    <xs:element ref="XSLTTRANSFORMATION" minOccurs="0"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="RELOADINITIALDATA">
  <xs:annotation>
   <xs:documentation>Reloads initial static data from the Context
Server.</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="GETDATALIST">
  <xs:annotation>
   <xs:documentation> get data elements (data to get from the CS).</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="GETDATA" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="RULE">
  <xs:annotation>
   <xs:documentation>Describes specific rules for specific data
received.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="CONDITION"/>
    <xs:element ref="ACTION"/>
   </xs:sequence>
   <xs:attribute name="Description" type="xs:string"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="XSLTTRANSFORMATION">
  <xs:annotation>
   <xs:documentation>XSL file to translate the incoming data if it matched the
rule.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:attribute name="file" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

7.2.3 Context Server Parameters

7.2.3.1 Categorization

```
<xs:element name="CONTEXT">
  <xs:annotation>
   <xs:documentation>Context of the Notification.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="SUBCONTEXT" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
```

```
 </xs:element>
 <xs:element name="SUBCONTEXT">
  <xs:annotation>
   <xs:documentation>Subcontext of the Context element.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="CATEGORY" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="SUBCONTEXTELEMENT" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="TYPE" use="required">
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="DYNAMIC"/>
      <xs:enumeration value="STATIC"/>
      <xs:enumeration value="NONE"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:complexType>
 </xs:element>
 <xs:element name="SUBCONTEXTELEMENT">
  <xs:annotation>
   <xs:documentation>Subcontext element.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="SUBCONTEXTELEMENT" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="NAME" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="CATEGORY">
  <xs:annotation>
   <xs:documentation>Category of the context.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:attribute name="TYPE" type="xs:string" use="required"/>
   <xs:attribute name="VALUE" type="xs:string" use="required"/>
  </xs:complexType>
</xs:element>
```

7.2.3.2 Subscribe

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:element name="SUBSCRIBE">
  <xs:annotation>
   <xs:documentation>Schema to subscribe at Context Server.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="PARAMETER" maxOccurs="unbounded">
     <xs:annotation>
      <xs:documentation>Parameter element defining subscription.</xs:documentation>
     </xs:annotation>
     <xs:complexType>
      <xs:simpleContent>
       <xs:extension base="xs:string">
        <xs:attribute name="CONDITION">
         <xs:annotation>
          <xs:documentation>Condition of the subscription.
ONCHANGE - sends data when there is a change in the subscribed data.
ONINTERVAL - sends data during the specified interval.</xs:documentation>
         </xs:annotation>
         <xs:simpleType>
          <xs:restriction base="xs:NMTOKEN">
           <xs:enumeration value="ONCHANGE"/>
           <xs:enumeration value="ONINTERVAL"/>
          </xs:restriction>
         </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="TIMES" type="xs:integer">
         <xs:annotation>
          <xs:documentation>The number of times the Context Server should send the
notification.</xs:documentation>
         </xs:annotation>
        </xs:attribute>
        <xs:attribute name="STARTTIME" type="xs:dateTime">
         <xs:annotation>
          <xs:documentation>The time when the Context Server should start sending
notifications.</xs:documentation>
         </xs:annotation>
        </xs:attribute>
        <xs:attribute name="ENDTIME" type="xs:dateTime">
         <xs:annotation>
          <xs:documentation>The time when the Context Server should stop sending
notifications.</xs:documentation>
         </xs:annotation>
        </xs:attribute>
        <xs:attribute name="INTERVAL" type="xs:integer">
         <xs:annotation>
          <xs:documentation>Time interval in which subscribed data is expected if the
condition is 'ONINTERVAL'.</xs:documentation>
         </xs:annotation>
```

```
        </xs:attribute>
       </xs:extension>
      </xs:simpleContent>
     </xs:complexType>
    </xs:element>
   </xs:sequence>
   <xs:attribute name="INFORMATIONTYPE" use="required">
    <xs:annotation>
     <xs:documentation>Defines the information type of the data. Can be either 'SDO' or
'Relation'.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="SDODATA"/>
      <xs:enumeration value="RELATION"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="SUBTREETYPE" use="required">
    <xs:annotation>
     <xs:documentation>Specifies either subscribing data changes only in the specied
context or changes including all its sub contexts.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="NODEONLY"/>
      <xs:enumeration value="SUBTREE"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="NAMESERVICEENTRY" type="xs:string" use="required">
    <xs:annotation>
     <xs:documentation>NameService name of the subscribing
component.</xs:documentation>
    </xs:annotation>
   </xs:attribute>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

7.2.3.3 Getdata

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:element name="GETDATA">
  <xs:annotation>
   <xs:documentation>Schema to get data from the Context Server.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="PARAMETER" type="xs:string" maxOccurs="unbounded"/>
   </xs:sequence>
   <xs:attribute name="DATATYPE" use="required">
    <xs:annotation>
     <xs:documentation>Type of data requested.
CONTEXTDATA - the data requested is from the Context or Relation profile.
HISTORYDATA - History data is requested.
SERVERNFORMATION - information about the Context server is
requested.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="CONTEXTDATA"/>
      <xs:enumeration value="HISTORYDATA"/>
      <xs:enumeration value="SERVERINFORMATION"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="INFORMATIONTYPE" use="required">
    <xs:annotation>
     <xs:documentation>Defines the information type of the data. Can be either 'SDO' or
'RELATION'.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="SDODATA"/>
      <xs:enumeration value="RELATION"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="SUBTREETYPE">
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="NODEONLY"/>
      <xs:enumeration value="SUBTREE"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="SERVERINFOTYPE">
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="GETCONTEXTPATH"/>
      <xs:enumeration value="GETHISTORYPATH"/>
      <xs:enumeration value="GETSERVERINFO"/>
     </xs:restriction>
    </xs:simpleType>
```

```
    </xs:attribute>
   </xs:complexType>
  </xs:element>
</xs:schema>
```

7.2.3.4 Notify

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified"
attributeFormDefault="unqualified">
 <xs:element name="NOTIFY">
  <xs:annotation>
   <xs:documentation>Notification to Context Server from SDO(s) or ICentric Services,
or vice versa.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:sequence>
     <xs:element ref="CONTEXT"/>
     <xs:element ref="ONTOLOGY"/>
     <xs:element ref="XMLDATA"/>
     <xs:element ref="SOURCE" minOccurs="0"/>
    </xs:sequence>
   </xs:sequence>
   <xs:attribute name="INFORMATIONTYPE" use="required">
    <xs:annotation>
     <xs:documentation>Defines the information type of the data. Can be either 'SDO' or
'Relation'.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="SDODATA"/>
      <xs:enumeration value="RELATION"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="CHANGETYPE" use="required">
    <xs:annotation>
     <xs:documentation>Intended action of the data. Can be 'Insert', 'modify',
'delete', or 'system' data.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="INSERTDATA"/>
      <xs:enumeration value="MODIFYDATA"/>
      <xs:enumeration value="DELETEDATA"/>
      <xs:enumeration value="SYSTEMDATA"/>
      <xs:enumeration value="REGISTERDATA"/>
      <xs:enumeration value="DEREGISTERDATA"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
   <xs:attribute name="SENDERID" type="xs:string" use="required">
    <xs:annotation>
     <xs:documentation>Name or ID of the sending component.</xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="UUID" type="xs:string">
    <xs:annotation>
     <xs:documentation>Unique ID of the data with the same categorization stored in the
Context Server.</xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="EVENTTIME" type="xs:dateTime">
    <xs:annotation>
     <xs:documentation>The time this event started. eg. the sensor detected an 'Actor'
in a 'Room'.</xs:documentation>
    </xs:annotation>
   </xs:attribute>
   <xs:attribute name="CSTIME" type="xs:dateTime">
    <xs:annotation>
     <xs:documentation>The time when the Context Server received the notification
message.</xs:documentation>
    </xs:annotation>
   </xs:attribute>
  </xs:complexType>
 </xs:element>
 <xs:element name="CONTEXT">
  <xs:annotation>
   <xs:documentation>Context of the Notification.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="SUBCONTEXT" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
 <xs:element name="SUBCONTEXT">
  <xs:annotation>
   <xs:documentation>Subcontext of the Context element.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="CATEGORY" minOccurs="0" maxOccurs="unbounded"/>
    <xs:element ref="SUBCONTEXTELEMENT" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="TYPE" use="required">
```

```
    <xs:annotation>
     <xs:documentation>Subcontext type.</xs:documentation>
    </xs:annotation>
    <xs:simpleType>
     <xs:restriction base="xs:NMTOKEN">
      <xs:enumeration value="DYNAMIC"/>
      <xs:enumeration value="STATIC"/>
      <xs:enumeration value="NONE"/>
     </xs:restriction>
    </xs:simpleType>
   </xs:attribute>
  </xs:complexType>
 </xs:element>
 <xs:element name="SUBCONTEXTELEMENT">
  <xs:annotation>
   <xs:documentation>Subcontext element.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element ref="SUBCONTEXTELEMENT" minOccurs="0"/>
   </xs:sequence>
   <xs:attribute name="NAME" type="xs:string" use="required"/>
  </xs:complexType>
 </xs:element>
 <xs:element name="CATEGORY">
  <xs:annotation>
   <xs:documentation>Category of the context.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:attribute name="TYPE" type="xs:string" use="required"/>
   <xs:attribute name="VALUE" type="xs:string" use="required">
    <xs:annotation>
     <xs:documentation>The value of the category type.</xs:documentation>
    </xs:annotation>
   </xs:attribute>
  </xs:complexType>
 </xs:element>
 <xs:element name="ONTOLOGY">
  <xs:annotation>
   <xs:documentation>Element defining Ontology of the sending
server.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:simpleContent>
    <xs:extension base="xs:string">
     <xs:attribute name="OID" type="xs:string" use="required">
      <xs:annotation>
       <xs:documentation>Unique ontology ID of the sending server.</xs:documentation>
      </xs:annotation>
     </xs:attribute>
    </xs:extension>
   </xs:simpleContent>
  </xs:complexType>
 </xs:element>
 <xs:element name="XMLDATA" type="xs:string">
  <xs:annotation>
   <xs:documentation>XML Data of the notification. Since the data can be of any
structure, it is inserted in the element as a string.</xs:documentation>
  </xs:annotation>
 </xs:element>
 <xs:element name="SOURCE">
  <xs:annotation>
   <xs:documentation>UUID(s) of the source data that create this relation
data.</xs:documentation>
  </xs:annotation>
  <xs:complexType>
   <xs:sequence>
    <xs:element name="UUID" type="xs:string" maxOccurs="unbounded"/>
   </xs:sequence>
  </xs:complexType>
 </xs:element>
</xs:schema>
```

7.3 Ontologies

7.3.1 Badge Sensor ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<ONTOLOGY xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
 <GENERAL>
  <ONTOLOGYTYPE>SDO</ONTOLOGYTYPE>
  <CONTEXT>/SDO/STATIC/DEVICE/PASSIVDEVICE</CONTEXT>
  <ONTOLOGYNAME>IRSENSOR</ONTOLOGYNAME>
  <VERSIONNUMBER>1.0</VERSIONNUMBER>
 </GENERAL>
 <CONTENT/>
 <USAGE>
  <OPERATION>
   <OPERATIONNAME>GetStatus</OPERATIONNAME>
   <OUTPARAMETER>
    <PARAMETERNAME>DeviceID</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:string"/>
```

```
      </xs:simpleType>
     </PARAMETERVALUE>
    </OUTPARAMETER>
    <OUTPARAMETER>
     <PARAMETERNAME>BadgeID</PARAMETERNAME>
     <PARAMETERVALUE>
      <xs:simpleType>
       <xs:restriction base="xs:string"/>
      </xs:simpleType>
     </PARAMETERVALUE>
    </OUTPARAMETER>
    <OUTPARAMETER>
     <PARAMETERNAME>TagID</PARAMETERNAME>
     <PARAMETERVALUE>
      <xs:simpleType>
       <xs:restriction base="xs:string"/>
      </xs:simpleType>
     </PARAMETERVALUE>
    </OUTPARAMETER>
    <OUTPARAMETER>
     <PARAMETERNAME>InMotion</PARAMETERNAME>
     <PARAMETERVALUE>
      <xs:simpleType>
       <xs:restriction base="xs:boolean"/>
      </xs:simpleType>
     </PARAMETERVALUE>
    </OUTPARAMETER>
    <OUTPARAMETER>
     <PARAMETERNAME>ButtonPressed</PARAMETERNAME>
     <PARAMETERVALUE>
      <xs:simpleType>
       <xs:restriction base="xs:boolean"/>
      </xs:simpleType>
     </PARAMETERVALUE>
    </OUTPARAMETER>
   </OPERATION>
  </USAGE>
  <MAPPINGS/>
  <DESCRIPTIONS>
   <DESCRIPTION>
    <LANGUAGE>EN</LANGUAGE>
    <LANGDESCRIPTION>badge sensor</LANGDESCRIPTION>
    <SHORTDESCRIPTION>badge sensor notifies about badges in its field of
view</SHORTDESCRIPTION>
    <PICTURE>badgesensor.jpg</PICTURE>
   </DESCRIPTION>
  </DESCRIPTIONS>
</ONTOLOGY>
```

7.3.2 Light Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<ONTOLOGY xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
  <GENERAL>
   <ONTOLOGYTYPE>SDO</ONTOLOGYTYPE>
   <CONTEXT>/SDO/STATIC/DEVICE/ACTIVEDEVICE</CONTEXT>
   <ONTOLOGYNAME>LIGHT</ONTOLOGYNAME>
   <VERSIONNUMBER>1.0</VERSIONNUMBER>
  </GENERAL>
  <CONTENT/>
  <USAGE>
   <OPERATION>
    <OPERATIONNAME>ChangeState</OPERATIONNAME>
    <INPARAMETER>
     <PARAMETERNAME>State</PARAMETERNAME>
     <PARAMETERVALUE>
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="on"/>
        <xs:enumeration value="off"/>
       </xs:restriction>
      </xs:simpleType>
     </PARAMETERVALUE>
    </INPARAMETER>
    <OUTPARAMETER>
     <PARAMETERNAME>State</PARAMETERNAME>
     <PARAMETERVALUE>
      <xs:simpleType>
       <xs:restriction base="xs:string">
        <xs:enumeration value="on"/>
        <xs:enumeration value="off"/>
       </xs:restriction>
      </xs:simpleType>
     </PARAMETERVALUE>
    </OUTPARAMETER>
   </OPERATION>
  </USAGE>
  <MAPPINGS/>
  <DESCRIPTIONS>
   <DESCRIPTION>
    <LANGUAGE>EN</LANGUAGE>
    <LANGDESCRIPTION>here you can describe the sdo</LANGDESCRIPTION>
    <SHORTDESCRIPTION>light</SHORTDESCRIPTION>
   </DESCRIPTION>
  </DESCRIPTIONS>
</ONTOLOGY>
```

7.3.3 Dimmer Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<ONTOLOGY xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
 <GENERAL>
  <ONTOLOGYTYPE>SDO</ONTOLOGYTYPE>
  <CONTEXT>/SDO/STATIC/DEVICE/ACTIVEDEVICE</CONTEXT>
  <ONTOLOGYNAME>DIMMER</ONTOLOGYNAME>
  <VERSIONNUMBER>1.0</VERSIONNUMBER>
 </GENERAL>
 <CONTENT/>
 <USAGE>
  <OPERATION>
   <OPERATIONNAME>Dimm</OPERATIONNAME>
   <INPARAMETER>
    <PARAMETERNAME>DimmValue</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:int">
       <xs:minInclusive value="0"/>
       <xs:maxInclusive value="100"/>
      </xs:restriction>
     </xs:simpleType>
    </PARAMETERVALUE>
   </INPARAMETER>
   <OUTPARAMETER>
    <PARAMETERNAME>Dimmed</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:int">
       <xs:minInclusive value="0"/>
       <xs:maxInclusive value="100"/>
      </xs:restriction>
     </xs:simpleType>
    </PARAMETERVALUE>
   </OUTPARAMETER>
  </OPERATION>
 </USAGE>
 <MAPPINGS>
  <MAPPING>
   <OPERATIONLINK>
    <CONTEXT>/SDO/STATIC/DEVICE/ACTIVEDEVICE/LIGHT/v1.0</CONTEXT>
    <OPERATIONNAME>ChangeState</OPERATIONNAME>
   </OPERATIONLINK>
   <TARGETOPERATIONNAME>Dimm</TARGETOPERATIONNAME>
   <MAPPINGINPARAMETER>
    <SOURCEPARAMETERNAME>State</SOURCEPARAMETERNAME>
    <TARGETPARAMETERNAME>DimmValue</TARGETPARAMETERNAME>
    <MAPPINGFUNCTION>{("off", "on"),("0", "100")}</MAPPINGFUNCTION>
   </MAPPINGINPARAMETER>
  </MAPPING>
 </MAPPINGS>
 <DESCRIPTIONS>
  <DESCRIPTION>
   <LANGUAGE>EN</LANGUAGE>
   <LANGDESCRIPTION>a dimmer can change the light intensity in a smooth
manner</LANGDESCRIPTION>
   <SHORTDESCRIPTION>dimmer</SHORTDESCRIPTION>
   <PICTURE>http://localhost:8080/media/Images/SDOimages/DIMMER_off.gif</PICTURE>
  </DESCRIPTION>
 </DESCRIPTIONS>
</ONTOLOGY>
```

7.3.4 PictureViewer Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<ONTOLOGY xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
 <GENERAL>
  <ONTOLOGYTYPE>SDO</ONTOLOGYTYPE>
  <CONTEXT>/SDO/STATIC/DEVICE/ACTIVEDEVICE</CONTEXT>
  <ONTOLOGYNAME>PICTUREVIEWER</ONTOLOGYNAME>
  <VERSIONNUMBER>1.0</VERSIONNUMBER>
 </GENERAL>
 <CONTENT/>
 <USAGE>
  <OPERATION>
   <OPERATIONNAME>Show</OPERATIONNAME>
   <INPARAMETER>
    <PARAMETERNAME>URL</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:string"/>
     </xs:simpleType>
    </PARAMETERVALUE>
   </INPARAMETER>
   <OUTPARAMETER>
    <PARAMETERNAME>Showing</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:string"/>
     </xs:simpleType>
    </PARAMETERVALUE>
   </OUTPARAMETER>
  </OPERATION>
 </USAGE>
```

```
 <MAPPINGS/>
 <DESCRIPTIONS>
  <DESCRIPTION>
   <LANGUAGE>EN</LANGUAGE>
   <LANGDESCRIPTION>PictureViewer, slide shows your pictures</LANGDESCRIPTION>
   <SHORTDESCRIPTION>PictureViewer</SHORTDESCRIPTION>
   <PICTURE>http://localhost:8080/media/Images/SDOimages/PictureViewer.gif</PICTURE>
  </DESCRIPTION>
 </DESCRIPTIONS>
</ONTOLOGY>
```

7.3.5 Aircondition Ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<ONTOLOGY xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
 <GENERAL>
  <ONTOLOGYTYPE>SDO</ONTOLOGYTYPE>
  <CONTEXT>/SDO/STATIC/DEVICE/ACTIVEDEVICE</CONTEXT>
  <ONTOLOGYNAME>AIRCONDITIONING</ONTOLOGYNAME>
  <VERSIONNUMBER>1.0</VERSIONNUMBER>
 </GENERAL>
 <CONTENT/>
 <USAGE>
  <OPERATION>
   <OPERATIONNAME>ChangeState</OPERATIONNAME>
   <INPARAMETER>
    <PARAMETERNAME>State</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="on"/>
       <xs:enumeration value="off"/>
      </xs:restriction>
     </xs:simpleType>
    </PARAMETERVALUE>
   </INPARAMETER>
   <OUTPARAMETER>
    <PARAMETERNAME>State</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:string">
       <xs:enumeration value="on"/>
       <xs:enumeration value="off"/>
      </xs:restriction>
     </xs:simpleType>
    </PARAMETERVALUE>
   </OUTPARAMETER>
  </OPERATION>
 </USAGE>
 <MAPPINGS/>
 <DESCRIPTIONS>
  <DESCRIPTION>
   <LANGUAGE>EN</LANGUAGE>
   <LANGDESCRIPTION>here you can describe the sdo</LANGDESCRIPTION>
   <SHORTDESCRIPTION>airconditioning</SHORTDESCRIPTION>

<PICTURE>http://localhost:8080/media/Images/SDOimages/AIRCONDITIONING_off.gif</PICTURE>
  </DESCRIPTION>
 </DESCRIPTIONS>
</ONTOLOGY>
```

7.3.6 Brightness Sensor ontology

```
<?xml version="1.0" encoding="UTF-8"?>
<ONTOLOGY xmlns:xs="http://www.w3.org/2001/XMLSchema"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance">
 <GENERAL>
  <ONTOLOGYTYPE>SDO</ONTOLOGYTYPE>
  <CONTEXT>/SDO/STATIC/DEVICE</CONTEXT>
  <ONTOLOGYNAME>BRIGHTNESSSENSOR</ONTOLOGYNAME>
  <VERSIONNUMBER>1.0</VERSIONNUMBER>
 </GENERAL>
 <CONTENT/>
 <USAGE>
  <OPERATION>
   <OPERATIONNAME>GetStatus</OPERATIONNAME>
   <OUTPARAMETER>
    <PARAMETERNAME>DeviceID</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:string"/>
     </xs:simpleType>
    </PARAMETERVALUE>
   </OUTPARAMETER>
   <OUTPARAMETER>
    <PARAMETERNAME>Brightness</PARAMETERNAME>
    <PARAMETERVALUE>
     <xs:simpleType>
      <xs:restriction base="xs:float">
       <xs:minInclusive value="0.0"/>
       <xs:maxInclusive value="100.0"/>
      </xs:restriction>
     </xs:simpleType>
    </PARAMETERVALUE>
   </OUTPARAMETER>
```

```
  </OPERATION>
 </USAGE>
 <MAPPINGS/>
 <DESCRIPTIONS>
  <DESCRIPTION>
   <LANGUAGE>EN</LANGUAGE>
   <LANGDESCRIPTION/>
   <SHORTDESCRIPTION>brightness sensor</SHORTDESCRIPTION>
   <PICTURE>brightness_sensor.jpg</PICTURE>
  </DESCRIPTION>
 </DESCRIPTIONS>
</ONTOLOGY>
```

7.4 XML / XSL Specification of realized Services

7.4.1 Customized Arbanowski (XML)

```
<?xml version="1.0" encoding="UTF-8"?>
<!--  ********************************************************
  ***      XML of ICentricService 'CustomizedArbanowski'     ***
  ********************************************************
  file:   CustomizedArbanowski.xml

  This is the configuration file for the above-mentioned ICentricService.
  It defines mandatory parameters and the overall behavior of the component.
-->
<CONFIGURATION>
 <!--
    #### General Parameter ####
 -->
 <PARAMETER NSOntologyServer=".ICCS.Ontology.OntologyServer"
NSContextServer=".ICCS.ContextServer.ContextServer" XSLT Executable="bin/TestXSLT.exe"
XSLT_Options="-Q -XML -IN %1 -XSL %2 -OUT %3" NumberOfThreads="5"/>
 <FILES TempPrefix="ICS_EveryLightOn_" TempFileDelete="false"
ParameterDTD="DTD/Parameter.dtd"/>
 <!-- Node to Registration the service at the Context Server-->
 <NOTIFY CHANGETYPE="INSERTDATA" INFORMATIONTYPE="SDODATA"
SENDERID="ICService4CusomizedArbanowski">
  <CONTEXT>
   <SUBCONTEXT TYPE="STATIC">
    <CATEGORY TYPE="Service" VALUE="CustomizedArbanowski"/>
    <SUBCONTEXTELEMENT NAME="SERVICE"/>
   </SUBCONTEXT>
  </CONTEXT>
  <ONTOLOGY OID="#initial_ontology#"/>
  <XMLDATA><![CDATA[
  <ICentricService name="CustomizedArbanowski">
   <Description value="CustomizedArbanowski Service"/>
  </ICentricService>
 ]]></XMLDATA>
 </NOTIFY>
 <!--
      #### Initial Data (static) that is preserved between actions of the component
####
 -->
 <INITIALSTATICDATA>
  <GETDATA DATATYPE="CONTEXTDATA" INFORMATIONTYPE="SDODATA" SUBTREETYPE="NODEONLY">

<PARAMETER>/DATA/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='SERVICE']/ancestor
::SUBCONTEXT</PARAMETER>

<PARAMETER>/DATA/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='LOCATION']/SUBCONT
EXTELEMENT[@NAME='ROOM']/ancestor::SUBCONTEXT</PARAMETER>

<PARAMETER>/DATA/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='ACTOR']/SUBCONTEXT
ELEMENT[@NAME='GENERAL']/ancestor::SUBCONTEXT</PARAMETER>

<PARAMETER>/DATA/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEX
TELEMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@NAME='JUKEBOX']/ancestor::SUBCONTEXT<
/PARAMETER>

<PARAMETER>/DATA/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEX
TELEMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@NAME='LIGHT']/ancestor::SUBCONTEXT</P
ARAMETER>

<PARAMETER>/DATA/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEX
TELEMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@NAME='PICTURE']/ancestor::SUBCONTEXT<
/PARAMETER>
  </GETDATA>
  <GETDATA DATATYPE="CONTEXTDATA" INFORMATIONTYPE="RELATION" SUBTREETYPE="NODEONLY">

<PARAMETER>/DATA/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='ACTOR']/SUBCONTEXT
ELEMENT[@NAME='SERVICE']/ancestor::SUBCONTEXT</PARAMETER>

<PARAMETER>/DATA/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='SERVICE']/SUBCONTE
XTELEMENT[@NAME='ACTOR']/ancestor::SUBCONTEXT</PARAMETER>

<PARAMETER>/DATA/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEX
TELEMENT[@NAME='LOCATION']/ancestor::SUBCONTEXT</PARAMETER>

<PARAMETER>/DATA/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='LOCATION']/SUBCONT
EXTELEMENT[@NAME='DEVICE']/ancestor::SUBCONTEXT</PARAMETER>
  </GETDATA>
 </INITIALSTATICDATA>
 <!--
```

```
      #### Contexts that have to be subscribed ####
 -->
 <SUBSCRIPTIONS>
  <SUBSCRIBE NAMESERVICEENTRY="" INFORMATIONTYPE="SDODATA" SUBTREETYPE="NODEONLY">
   <PARAMETER>STATIC/LOCATION/ROOM</PARAMETER>
   <PARAMETER>STATIC/Actor/General</PARAMETER>
   <PARAMETER>STATIC/Device/ActiveDevice/Light</PARAMETER>
   <PARAMETER>STATIC/Service</PARAMETER>
  </SUBSCRIBE>
  <SUBSCRIBE NAMESERVICEENTRY="" INFORMATIONTYPE="RELATION" SUBTREETYPE="NODEONLY">
   <PARAMETER>DYNAMIC/Actor/Location</PARAMETER>
   <PARAMETER>STATIC/Device/Location</PARAMETER>
   <PARAMETER>STATIC/Actor/Service</PARAMETER>
  </SUBSCRIBE>
 </SUBSCRIPTIONS>
 <!--
      #### Action-Rules for Notifications ####
 -->
 <BEHAVIOR>
  <RULE Description="Reload Static Data if it is changed!">
   <CONDITION InformationType="ANY" ChangeType="ANY">
    <CONTEXT>
     <SUBCONTEXT TYPE="STATIC"/>
    </CONTEXT>
   </CONDITION>
   <ACTION>
    <RELOADINITIALDATA/>
   </ACTION>
  </RULE>
  <RULE Description="Actual Work will be done with this Rule!">
   <CONDITION InformationType="RELATION" ChangeType="ANY">
    <CONTEXT>
     <SUBCONTEXT TYPE="DYNAMIC">
      <SUBCONTEXTELEMENT NAME="Actor">
       <SUBCONTEXTELEMENT NAME="Location"/>
      </SUBCONTEXTELEMENT>
     </SUBCONTEXT>
    </CONTEXT>
   </CONDITION>
   <ACTION>
    <GETDATALIST>
     <GETDATA DATATYPE="CONTEXTDATA" INFORMATIONTYPE="RELATION" SUBTREETYPE="NODEONLY">
      <PARAMETER>/DATA/SUBCONTEXT[@TYPE='DYNAMIC'][(SUBCONTEXTELEMENT/@NAME='ACTOR' and
SUBCONTEXTELEMENT/SUBCONTEXTELEMENT/@NAME='LOCATION') or
(SUBCONTEXTELEMENT/@NAME='LOCATION' and
SUBCONTEXTELEMENT/SUBCONTEXTELEMENT/@NAME='ACTOR')]</PARAMETER>
     </GETDATA>
     <GETDATA DATATYPE="CONTEXTDATA" INFORMATIONTYPE="SDODATA" SUBTREETYPE="NODEONLY">
      <PARAMETER>/DATA/SUBCONTEXT[@TYPE='DYNAMIC'][(SUBCONTEXTELEMENT/@NAME='SYSTEM'
and SUBCONTEXTELEMENT/SUBCONTEXTELEMENT/@NAME='ICENTRICSERVICE' and
SUBCONTEXTELEMENT/SUBCONTEXTELEMENT/SUBCONTEXTELEMENT/@NAME='REQUEST')]</PARAMETER>
     </GETDATA>
    </GETDATALIST>
    <XSLTTRANSFORMATION file="Configuration/ICentricService/CustomizedArbanowski.xsl"/>
   </ACTION>
  </RULE>
 </BEHAVIOR>
</CONFIGURATION>
```

7.4.2 Customized Arbanowski (XSL)

```
<?xml version="1.0" encoding="UTF-8"?>
<!--  ************************************************************
  ***    XSL of ICentricService 'CustomizedArbanowski'    ***
  ************************************************************
  file:   CustomizedArbanowski.xsl

  This ICentricService is fully cusomized for Stefan Arbansowski. The LightSDO,
JukeboxSDO and  PictureSDO are controlled in
  this XSL stylesheet by using predefined values. These values are stored within the
ICService, so if the user want to have another behavior, the ICService has to be
  adapted.
-->
<xsl:stylesheet version="1" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
xmlns:rm="file:/ContextServer/schema/RelationMap#"
xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:saxon="http://icl.com/saxon" extension-element-prefixes="saxon">
 <!-- ************************************************************
    Variable - Definition
 ************************************************************ -->
 <xsl:variable name="ownServiceName">
  <xsl:text>CustomizedArbanowski</xsl:text>
 </xsl:variable>
 <xsl:variable name="ownServiceUUID">
  <xsl:value-of select="/ROOT/SUBCONTEXT[@TYPE='STATIC']/CATEGORY[@VALUE =
$ownServiceName]/../SUBCONTEXTELEMENT[@NAME='SERVICE']/DATAELEMENT/@rdf:about"/>
 </xsl:variable>
 <xsl:variable name="nameRelDevLoc">
  <xsl:text>DEVICE in LOCATION</xsl:text>
 </xsl:variable>
 <xsl:variable name="nameRelActLoc">
  <xsl:text>ACTOR in LOCATION</xsl:text>
 </xsl:variable>
 <xsl:variable name="actorUUID">
```

```
  <xsl:value-of
select="/ROOT/NOTIFYDATAELEMENT/XMLDATA/rdf:RDF/rm:Relation/rm:SDODataReference/rm:SDOD
ataContext/CONTEXT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='ACTOR']/SUBCONTE
XTELEMENT[@NAME='GENERAL']/ancestor::rm:SDODataReference/rm:SDODataUUID/@rdf:resource"/
>
 </xsl:variable>
 <xsl:variable name="locationUUID">
  <xsl:value-of
select="/ROOT/NOTIFYDATAELEMENT/XMLDATA/rdf:RDF/rm:Relation/rm:SDODataReference/rm:SDOD
ataContext/CONTEXT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='LOCATION']/SUBCO
NTEXTELEMENT[@NAME='ROOM']/ancestor::rm:SDODataReference/rm:SDODataUUID/@rdf:resource"/
>
 </xsl:variable>
 <xsl:variable name="locationID">
  <xsl:value-of
select="/ROOT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='LOCATION']/SUBCONTEXT
ELEMENT[@NAME='ROOM']/DATAELEMENT[@rdf:about = $locationUUID]/XMLDATA/Location/@ID"/>
 </xsl:variable>
 <xsl:variable name="actorID">
  <xsl:value-of
select="/ROOT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='ACTOR']/SUBCONTEXTELE
MENT[@NAME='GENERAL']/DATAELEMENT[@rdf:about = $actorUUID]/XMLDATA/Actor/@ID"/>
 </xsl:variable>
 <!-- ************************************************************
    ROOT - Template

    Description: Main processing template
 ************************************************************ -->
 <xsl:template match="ROOT">
  <OUTPUT>
   <xsl:text>
   ownServiceName: </xsl:text>
   <xsl:value-of select="$ownServiceName"/>
   <xsl:text>
   ownServiceUUID: </xsl:text>
   <xsl:value-of select="$ownServiceUUID"/>
   <xsl:text>
   actorUUID: </xsl:text>
   <xsl:value-of select="$actorUUID"/>
   <xsl:text>
   actorID: </xsl:text>
   <xsl:value-of select="$actorID"/>
   <xsl:text>
   locationUUID: </xsl:text>
   <xsl:value-of select="$locationUUID"/>
   <xsl:text>
   locationID: </xsl:text>
   <xsl:value-of select="$locationID"/>
   <xsl:text>
   </xsl:text>
   <!--
      Check if there is an existing Request which should be executed  (ONLY When
DELETEDATA or MODIFYDATA)
      (-> Get all requests for this Service and see if the conditions are true.)
   -->
   <xsl:if test="(NOTIFYDATAELEMENT/@changeType = 'MODIFYDATA') or
(NOTIFYDATAELEMENT/@changeType = 'DELETEDATA')">
    <xsl:apply-templates
select="SUBCONTEXT[@TYPE='DYNAMIC']/CATEGORY[(@TYPE='ICentricService') and
(@VALUE=$ownServiceName)]/../SUBCONTEXTELEMENT[@NAME='SYSTEM']/SUBCONTEXTELEMENT[@NAME=
'ICENTRICSERVICE']/SUBCONTEXTELEMENT[@NAME='REQUEST']/DATAELEMENT" mode="request"/>
   </xsl:if>
   <!--
      Should the light be switched on? Check the following:
      1. IF :  is Notify-changeType == INSERTDATA or MODIFYDATA
      2. IF :  is the Notify-Relation the right on and is there a subscription
         by the actor for this Service?
   -->

   <xsl:if test="(NOTIFYDATAELEMENT/@changeType = 'MODIFYDATA') or
(NOTIFYDATAELEMENT/@changeType = 'INSERTDATA')">


    <!-- Does the Actor have a subsciption with service -->
    <xsl:if test="(NOTIFYDATAELEMENT/XMLDATA/rdf:RDF/rm:Relation/@name=$nameRelActLoc)
and
(SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='ACTOR']/SUBCONTEXTELEMENT[@NAME =
'SERVICE']/DATAELEMENT/XMLDATA/descendant::SUBCONTEXTELEMENT[@NAME =
'ACTOR']/ancestor::rm:SDODataReference/rm:SDODataUUID[@rdf:resource =
$actorUUID]/ancestor::XMLDATA/descendant::SUBCONTEXTELEMENT[@NAME =
'SERVICE']/ancestor::rm:SDODataReference/rm:SDODataUUID/@rdf:resource =
$ownServiceUUID)">

     <!-- FOR EACH device in a relation with the location -->
     <xsl:for-each
select="/ROOT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEXTEL
EMENT[@NAME='LOCATION']/DATAELEMENT/XMLDATA/rdf:RDF/rm:Relation[@name=$nameRelDevLoc]">
      <xsl:if
test="rm:SDODataReference/rm:SDODataContext/CONTEXT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTE
XTELEMENT[@NAME='LOCATION']/ancestor::rm:SDODataReference/rm:SDODataUUID/@rdf:resource
= $locationUUID">
       <!-- find deviceUUID for light-devices in this location-->
       <xsl:variable name="tempDeviceUUID" saxon:assignable="yes">
        <xsl:value-of
select="rm:SDODataReference/rm:SDODataContext/CONTEXT/SUBCONTEXT[@TYPE='STATIC']/SUBCON
TEXTELEMENT[@NAME='DEVICE']/SUBCONTEXTELEMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@
NAME='LIGHT']/ancestor::rm:SDODataReference/rm:SDODataUUID/@rdf:resource"/>
       </xsl:variable>
       <!-- if there is a device, activate it -->
```

```
       <xsl:if test="(string-length($tempDeviceUUID) > 0)">
        <NOTIFY INFORMATIONTYPE="SDODATA" CHANGETYPE="MODIFYDATA">
         <xsl:attribute name="SENDERID"><xsl:value-of
select="$ownServiceName"/></xsl:attribute>
         <xsl:apply-templates
select="/ROOT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEXTEL
EMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@NAME='LIGHT']/DATAELEMENT[@rdf:about =
$tempDeviceUUID]" mode="on"/>
        </NOTIFY>
        <!-- and now set a request, that the light should be turned off, when the actor
leaves the room -->
        <NOTIFY INFORMATIONTYPE="SDODATA" CHANGETYPE="MODIFYDATA">
         <xsl:attribute name="SENDERID"><xsl:value-of
select="$ownServiceName"/></xsl:attribute>
         <CONTEXT>
          <!-- SUBCONTEXT: /SYSTEM/ICENTRICSERVICE/REQUEST -->
          <SUBCONTEXT TYPE="DYNAMIC">
           <CATEGORY TYPE="ICentricService" VALUE="{$ownServiceName}"/>
           <CATEGORY TYPE="ActorID" VALUE="{$actorID}"/>
           <CATEGORY TYPE="LocationID" VALUE="{$locationID}"/>
           <CATEGORY TYPE="DeviceID" VALUE="{$tempDeviceUUID}"/>
           <SUBCONTEXTELEMENT NAME="SYSTEM">
            <SUBCONTEXTELEMENT NAME="ICENTRICSERVICE">
             <SUBCONTEXTELEMENT NAME="REQUEST"/>
            </SUBCONTEXTELEMENT>
           </SUBCONTEXTELEMENT>
          </SUBCONTEXT>
         </CONTEXT>
         <ONTOLOGY OID="#ICS-ontology: CustomizedArbanowski Request">
      The OID is just for testing purposes only!
         </ONTOLOGY>
         <XMLDATA>
          <Condition>
           <RelationNotExist>
            <xsl:attribute name="rdf:resource"><xsl:value-of
select="/ROOT/NOTIFYDATAELEMENT/@UUID"/></xsl:attribute>
            <xsl:apply-templates select="/ROOT/NOTIFYDATAELEMENT/*" mode="copy"/>
           </RelationNotExist>
          </Condition>
          <Action>
           <!-- wrapp the NOTIFY-part in a doNOTIFY-Tag. Otherwise the ICentricService-
exe
            does not know, which NOTIFY is a real one and which is only included in a
request.. -->
           <doNOTIFY INFORMATIONTYPE="SDODATA" CHANGETYPE="MODIFYDATA">
            <xsl:attribute name="SENDERID"><xsl:value-of
select="$ownServiceName"/></xsl:attribute>
            <xsl:apply-templates
select="/ROOT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEXTEL
EMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@NAME='LIGHT']/DATAELEMENT[@rdf:about =
$tempDeviceUUID]" mode="off"/>
           </doNOTIFY>
          </Action>
         </XMLDATA>
        </NOTIFY>
       </xsl:if>


       <!-- find device UUID for all jukebox-devices -->
       <saxon:assign name="tempDeviceUUID">
        <xsl:value-of
select="rm:SDODataReference/rm:SDODataContext/CONTEXT/SUBCONTEXT[@TYPE='STATIC']/SUBCON
TEXTELEMENT[@NAME='DEVICE']/SUBCONTEXTELEMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@
NAME='JUKEBOX']/ancestor::rm:SDODataReference/rm:SDODataUUID/@rdf:resource"/>
       </saxon:assign>
       <!-- if there is a device, activate it -->
       <xsl:if test="(string-length($tempDeviceUUID) > 0)">
        <NOTIFY INFORMATIONTYPE="SDODATA" CHANGETYPE="MODIFYDATA">
         <xsl:attribute name="SENDERID"><xsl:value-of
select="$ownServiceName"/></xsl:attribute>
         <xsl:apply-templates
select="/ROOT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEXTEL
EMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@NAME='JUKEBOX']/DATAELEMENT[@rdf:about =
$tempDeviceUUID]" mode="play"/>
        </NOTIFY>
        <!-- and now set a request, that the light should be turned off, when the actor
leaves the room -->
        <NOTIFY INFORMATIONTYPE="SDODATA" CHANGETYPE="MODIFYDATA">
         <xsl:attribute name="SENDERID"><xsl:value-of
select="$ownServiceName"/></xsl:attribute>
         <CONTEXT>
          <!-- SUBCONTEXT: /SYSTEM/ICENTRICSERVICE/REQUEST -->
          <SUBCONTEXT TYPE="DYNAMIC">
           <CATEGORY TYPE="ICentricService" VALUE="{$ownServiceName}"/>
           <CATEGORY TYPE="ActorID" VALUE="{$actorID}"/>
           <CATEGORY TYPE="LocationID" VALUE="{$locationID}"/>
           <CATEGORY TYPE="DeviceID" VALUE="{$tempDeviceUUID}"/>
           <SUBCONTEXTELEMENT NAME="SYSTEM">
            <SUBCONTEXTELEMENT NAME="ICENTRICSERVICE">
             <SUBCONTEXTELEMENT NAME="REQUEST"/>
            </SUBCONTEXTELEMENT>
           </SUBCONTEXTELEMENT>
          </SUBCONTEXT>
         </CONTEXT>
         <ONTOLOGY OID="#ICS-ontology: CustomizedArbanowski Request">
      The OID is just for testing purposes only!
         </ONTOLOGY>
         <XMLDATA>
          <Condition>
```

```
            <RelationNotExist>
             <xsl:attribute name="rdf:resource"><xsl:value-of
select="/ROOT/NOTIFYDATAELEMENT/@UUID"/></xsl:attribute>
             <xsl:apply-templates select="/ROOT/NOTIFYDATAELEMENT/*" mode="copy"/>
            </RelationNotExist>
           </Condition>
           <Action>
            <!-- wrapp the NOTIFY-part in a doNOTIFY-Tag. Otherwise the ICentricService-
exe
             does not know, which NOTIFY is a real one and which is only included in a
request.. -->
            <doNOTIFY INFORMATIONTYPE="SDODATA" CHANGETYPE="MODIFYDATA">
             <xsl:attribute name="SENDERID"><xsl:value-of
select="$ownServiceName"/></xsl:attribute>
             <xsl:apply-templates
select="/ROOT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEXTEL
EMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@NAME='JUKEBOX']/DATAELEMENT[@rdf:about =
$tempDeviceUUID]" mode="stop"/>
            </doNOTIFY>
           </Action>
          </XMLDATA>
         </NOTIFY>
        </xsl:if>

        <!-- find device UUID for all picture-devices -->
        <saxon:assign name="tempDeviceUUID">
         <xsl:value-of
select="rm:SDODataReference/rm:SDODataContext/CONTEXT/SUBCONTEXT[@TYPE='STATIC']/SUBCON
TEXTELEMENT[@NAME='DEVICE']/SUBCONTEXTELEMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@
NAME='PICTURE']/ancestor::rm:SDODataReference/rm:SDODataUUID/@rdf:resource"/>
</saxon:assign>
        <!-- if there is a picture device, activate it -->
        <xsl:if test="(string-length($tempDeviceUUID) > 0)">
         <NOTIFY INFORMATIONTYPE="SDODATA" CHANGETYPE="MODIFYDATA">
          <xsl:attribute name="SENDERID"><xsl:value-of
select="$ownServiceName"/></xsl:attribute>
           <xsl:apply-templates
select="/ROOT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEXTEL
EMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@NAME='PICTURE']/DATAELEMENT[@rdf:about =
$tempDeviceUUID]" mode="show"/>
         </NOTIFY>
         <!-- and now set a request, that the light should be turned off, when the actor
leaves the room -->
         <NOTIFY INFORMATIONTYPE="SDODATA" CHANGETYPE="MODIFYDATA">
          <xsl:attribute name="SENDERID"><xsl:value-of
select="$ownServiceName"/></xsl:attribute>
           <CONTEXT>
            <!-- SUBCONTEXT: /SYSTEM/ICENTRICSERVICE/REQUEST -->
            <SUBCONTEXT TYPE="DYNAMIC">
             <CATEGORY TYPE="ICentricService" VALUE="{$ownServiceName}"/>
             <CATEGORY TYPE="ActorID" VALUE="{$actorID}"/>
             <CATEGORY TYPE="LocationID" VALUE="{$locationID}"/>
             <CATEGORY TYPE="DeviceID" VALUE="{$tempDeviceUUID}"/>
             <SUBCONTEXTELEMENT NAME="SYSTEM">
              <SUBCONTEXTELEMENT NAME="ICENTRICSERVICE">
               <SUBCONTEXTELEMENT NAME="REQUEST"/>
              </SUBCONTEXTELEMENT>
             </SUBCONTEXTELEMENT>
            </SUBCONTEXT>
           </CONTEXT>
           <ONTOLOGY OID="#ICS-ontology: CustomizedArbanowski Request">
      The OID is just for testing purposes only!
           </ONTOLOGY>
           <XMLDATA>
            <Condition>
             <RelationNotExist>
              <xsl:attribute name="rdf:resource"><xsl:value-of
select="/ROOT/NOTIFYDATAELEMENT/@UUID"/></xsl:attribute>
              <xsl:apply-templates select="/ROOT/NOTIFYDATAELEMENT/*" mode="copy"/>
             </RelationNotExist>
            </Condition>
            <Action>
             <!-- wrapp the NOTIFY-part in a doNOTIFY-Tag. Otherwise the ICentricService-
exe
              does not know, which NOTIFY is a real one and which is only included in a
request.. -->
             <doNOTIFY INFORMATIONTYPE="SDODATA" CHANGETYPE="MODIFYDATA">
              <xsl:attribute name="SENDERID"><xsl:value-of
select="$ownServiceName"/></xsl:attribute>
              <xsl:apply-templates
select="/ROOT/SUBCONTEXT[@TYPE='STATIC']/SUBCONTEXTELEMENT[@NAME='DEVICE']/SUBCONTEXTEL
EMENT[@NAME='ACTIVEDEVICE']/SUBCONTEXTELEMENT[@NAME='PICTURE']/DATAELEMENT[@rdf:about =
$tempDeviceUUID]" mode="endshow"/>
             </doNOTIFY>
            </Action>
           </XMLDATA>
          </NOTIFY>
         </xsl:if>
        </xsl:if>
       </xsl:for-each>
      </xsl:if>
     </xsl:if>
    </OUTPUT>
   </xsl:template>
   <!-- ********************************************************
      DATAELEMENT- Template - mode = on

      Description:  writes the activation for the
           DATAELEMENT-device to the output XML
```

```
 ************************************************************ -->
 <xsl:template match="DATAELEMENT" mode="on">
  <CONTEXT>
   <!-- SUBCONTEXT: /SYSTEM/REQUEST/DEVICE -->
   <SUBCONTEXT TYPE="DYNAMIC">
    <CATEGORY TYPE="ICentricService" VALUE="{$ownServiceName}"/>
    <CATEGORY TYPE="ActorID" VALUE="{$actorID}"/>
    <CATEGORY TYPE="LocationID" VALUE="{$locationID}"/>
    <CATEGORY TYPE="DeviceID" VALUE="{@rdf:about}"/>
    <SUBCONTEXTELEMENT NAME="SYSTEM">
     <SUBCONTEXTELEMENT NAME="REQUEST">
      <SUBCONTEXTELEMENT NAME="DEVICE"/>
     </SUBCONTEXTELEMENT>
    </SUBCONTEXTELEMENT>
   </SUBCONTEXT>
  </CONTEXT>
  <ONTOLOGY OID="#ICS-ontology: Device Request">
      The OID is just for testing purposes only!
     </ONTOLOGY>
  <XMLDATA>
   <Request>
    <ActiveDevice>
     <xsl:attribute name="rdf:resource"><xsl:value-of
select="@rdf:about"/></xsl:attribute>
    </ActiveDevice>
    <EXECUTE>
     <OPERATION NAME="ChangeState">
      <PARAMETER NAME="State" VALUE="on"/>
     </OPERATION>
    </EXECUTE>
   </Request>
  </XMLDATA>
 </xsl:template>
 <!-- ************************************************************
    DATAELEMENT- Template - mode = off

    Description:  writes the deactivation for the
         DATAELEMENT-device to the output XML
 ************************************************************ -->
 <xsl:template match="DATAELEMENT" mode="off">
  <CONTEXT>
   <!-- SUBCONTEXT: /SYSTEM/REQUEST/DEVICE -->
   <SUBCONTEXT TYPE="DYNAMIC">
    <CATEGORY TYPE="ICentricService" VALUE="{$ownServiceName}"/>
    <CATEGORY TYPE="ActorID" VALUE="{$actorID}"/>
    <CATEGORY TYPE="LocationID" VALUE="{$locationID}"/>
    <CATEGORY TYPE="DeviceID" VALUE="{@rdf:about}"/>
    <SUBCONTEXTELEMENT NAME="SYSTEM">
     <SUBCONTEXTELEMENT NAME="REQUEST">
      <SUBCONTEXTELEMENT NAME="DEVICE"/>
     </SUBCONTEXTELEMENT>
    </SUBCONTEXTELEMENT>
   </SUBCONTEXT>
  </CONTEXT>
  <ONTOLOGY OID="#ICS-ontology: Device Request">
      The OID is just for testing purposes only!
     </ONTOLOGY>
  <XMLDATA>
   <Request>
    <ActiveDevice>
     <xsl:attribute name="rdf:resource"><xsl:value-of
select="@rdf:about"/></xsl:attribute>
    </ActiveDevice>
    <EXECUTE>
     <OPERATION NAME="ChangeState">
      <PARAMETER NAME="State" VALUE="off"/>
     </OPERATION>
    </EXECUTE>
   </Request>
  </XMLDATA>
 </xsl:template>
 <!-- ************************************************************
    DATAELEMENT- Template - mode = play

    Description:  write the notify
 ************************************************************ -->
 <xsl:template match="DATAELEMENT" mode="play">
  <CONTEXT>
   <!-- SUBCONTEXT: /SYSTEM/REQUEST/DEVICE -->
   <SUBCONTEXT TYPE="DYNAMIC">
    <CATEGORY TYPE="ICentricService" VALUE="{$ownServiceName}"/>
    <CATEGORY TYPE="ActorID" VALUE="{$actorID}"/>
    <CATEGORY TYPE="LocationID" VALUE="{$locationID}"/>
    <CATEGORY TYPE="DeviceID" VALUE="{@rdf:about}"/>
    <SUBCONTEXTELEMENT NAME="SYSTEM">
     <SUBCONTEXTELEMENT NAME="REQUEST">
      <SUBCONTEXTELEMENT NAME="DEVICE"/>
     </SUBCONTEXTELEMENT>
    </SUBCONTEXTELEMENT>
   </SUBCONTEXT>
  </CONTEXT>
  <ONTOLOGY OID="#ICS-ontology: Device Request">
      The OID is just for testing purposes only!
     </ONTOLOGY>
  <XMLDATA>
   <Request>
    <ActiveDevice>
     <xsl:attribute name="rdf:resource"><xsl:value-of
select="@rdf:about"/></xsl:attribute>
```

```
    </ActiveDevice>
    <EXECUTE>
     <OPERATION NAME="play">
      <PARAMETER NAME="file" VALUE="StefanArbanowski"/>
     </OPERATION>
    </EXECUTE>
   </Request>
  </XMLDATA>
 </xsl:template>
 <!-- ************************************************************
    DATAELEMENT- Template - mode = stop

    Description:  write the notify
 ********************************************************** -->
 <xsl:template match="DATAELEMENT" mode="stop">
  <CONTEXT>
   <!-- SUBCONTEXT: /SYSTEM/REQUEST/DEVICE -->
   <SUBCONTEXT TYPE="DYNAMIC">
    <CATEGORY TYPE="ICentricService" VALUE="{$ownServiceName}"/>
    <CATEGORY TYPE="ActorID" VALUE="{$actorID}"/>
    <CATEGORY TYPE="LocationID" VALUE="{$locationID}"/>
    <CATEGORY TYPE="DeviceID" VALUE="{@rdf:about}"/>
    <SUBCONTEXTELEMENT NAME="SYSTEM">
     <SUBCONTEXTELEMENT NAME="REQUEST">
      <SUBCONTEXTELEMENT NAME="DEVICE"/>
     </SUBCONTEXTELEMENT>
    </SUBCONTEXTELEMENT>
   </SUBCONTEXT>
  </CONTEXT>
  <ONTOLOGY OID="#ICS-ontology: Device Request">
      The OID is just for testing purposes only!
     </ONTOLOGY>
  <XMLDATA>
   <Request>
    <ActiveDevice>
     <xsl:attribute name="rdf:resource"><xsl:value-of
select="@rdf:about"/></xsl:attribute>
    </ActiveDevice>
    <EXECUTE>
     <OPERATION NAME="stop"/>
    </EXECUTE>
   </Request>
  </XMLDATA>
 </xsl:template>

 <!-- ************************************************************
    DATAELEMENT- Template - mode = show

    Description:  write the notify
 ********************************************************** -->
 <xsl:template match="DATAELEMENT" mode="show">
  <CONTEXT>
   <!-- SUBCONTEXT: /SYSTEM/REQUEST/DEVICE -->
   <SUBCONTEXT TYPE="DYNAMIC">
    <CATEGORY TYPE="ICentricService" VALUE="{$ownServiceName}"/>
    <CATEGORY TYPE="ActorID" VALUE="{$actorID}"/>
    <CATEGORY TYPE="LocationID" VALUE="{$locationID}"/>
    <CATEGORY TYPE="DeviceID" VALUE="{@rdf:about}"/>
    <SUBCONTEXTELEMENT NAME="SYSTEM">
     <SUBCONTEXTELEMENT NAME="REQUEST">
      <SUBCONTEXTELEMENT NAME="DEVICE"/>
     </SUBCONTEXTELEMENT>
    </SUBCONTEXTELEMENT>
   </SUBCONTEXT>
  </CONTEXT>
  <ONTOLOGY OID="#ICS-ontology: Device Request">
      The OID is just for testing purposes only!
     </ONTOLOGY>
  <XMLDATA>
   <Request>
    <ActiveDevice>
     <xsl:attribute name="rdf:resource"><xsl:value-of
select="@rdf:about"/></xsl:attribute>
    </ActiveDevice>
    <EXECUTE>
     <OPERATION NAME="Show">
      <PARAMETER NAME="URL" VALUE="StefanArbanowski"/>
     </OPERATION>
    </EXECUTE>
   </Request>
  </XMLDATA>
 </xsl:template>
 <!-- ************************************************************
    DATAELEMENT- Template - mode = endshow

    Description:  write the notify
 ********************************************************** -->
 <xsl:template match="DATAELEMENT" mode="endshow">
  <CONTEXT>
   <!-- SUBCONTEXT: /SYSTEM/REQUEST/DEVICE -->
   <SUBCONTEXT TYPE="DYNAMIC">
    <CATEGORY TYPE="ICentricService" VALUE="{$ownServiceName}"/>
    <CATEGORY TYPE="ActorID" VALUE="{$actorID}"/>
    <CATEGORY TYPE="LocationID" VALUE="{$locationID}"/>
    <CATEGORY TYPE="DeviceID" VALUE="{@rdf:about}"/>
    <SUBCONTEXTELEMENT NAME="SYSTEM">
     <SUBCONTEXTELEMENT NAME="REQUEST">
      <SUBCONTEXTELEMENT NAME="DEVICE"/>
     </SUBCONTEXTELEMENT>
```

```
    </SUBCONTEXTELEMENT>
   </SUBCONTEXT>
  </CONTEXT>
  <ONTOLOGY OID="#ICS-ontology: Device Request">
      The OID is just for testing purposes only!
    </ONTOLOGY>
  <XMLDATA>
   <Request>
    <ActiveDevice>
     <xsl:attribute name="rdf:resource"><xsl:value-of
select="@rdf:about"/></xsl:attribute>
    </ActiveDevice>
    <EXECUTE>
     <OPERATION NAME="Show">
      <PARAMETER NAME="URL" VALUE="General"/>
     </OPERATION>
    </EXECUTE>
   </Request>
  </XMLDATA>
 </xsl:template>

 <!-- ********************************************************
    doNOTIFY- Template

    Description:  unwrappes the NOTIFY-Part in a request
          to a real NOTIFY in the output XML
 ******************************************************** -->
 <xsl:template match="doNOTIFY">
  <NOTIFY>
   <xsl:attribute name="INFORMATIONTYPE"><xsl:value-of
select="@INFORMATIONTYPE"/></xsl:attribute>
   <xsl:attribute name="CHANGETYPE"><xsl:value-of
select="@CHANGETYPE"/></xsl:attribute>
   <xsl:attribute name="SENDERID"><xsl:value-of select="@SENDERID"/></xsl:attribute>
   <xsl:apply-templates select="*" mode="copy"/>
  </NOTIFY>
 </xsl:template>
 <!-- ********************************************************
    DATAELEMENT- Template - mode = request

    Description:  processes a request
          (check condition / execute action)
 ******************************************************** -->
 <xsl:template match="DATAELEMENT" mode="request">
  <xsl:variable name="tempUUID">
   <xsl:value-of select="XMLDATA/Condition/RelationNotExist/@rdf:resource"/>
  </xsl:variable>
  <!-- is it time to perform the request? check if the relation still exists. -->
  <xsl:if test="not(
/ROOT/SUBCONTEXT[@TYPE='DYNAMIC']/SUBCONTEXTELEMENT[@NAME='ACTOR']/SUBCONTEXTELEMENT[@N
AME='LOCATION']/DATAELEMENT[@rdf:about = $tempUUID]/@rdf:about = $tempUUID )">
   <!-- do what the request has in its action-part -->
   <xsl:apply-templates select="XMLDATA/Action/doNOTIFY"/>
   <!-- and now delete this already performed request -->
   <NOTIFY INFORMATIONTYPE="SDODATA" CHANGETYPE="DELETEDATA">
    <xsl:attribute name="SENDERID"><xsl:value-of
select="$ownServiceName"/></xsl:attribute>
    <CONTEXT>
     <!-- SUBCONTEXT: /SYSTEM/ICENTRICSERVICE/REQUEST -->
     <SUBCONTEXT TYPE="DYNAMIC">
      <CATEGORY TYPE="ICentricService" VALUE="{$ownServiceName}"/>
      <CATEGORY TYPE="ActorID">
       <xsl:attribute name="VALUE"><xsl:value-of
select="CONTEXT/SUBCONTEXT[@TYPE='DYNAMIC']/SUBCONTEXTELEMENT[@NAME='SYSTEM']/SUBCONTEX
TELEMENT[@NAME='ICENTRICSERVICE']/SUBCONTEXTELEMENT[@NAME='REQUEST']/ancestor::SUBCONTE
XT/CATEGORY[@TYPE='ActorID']/@VALUE"/></xsl:attribute>
      </CATEGORY>
      <CATEGORY TYPE="LocationID">
       <xsl:attribute name="VALUE"><xsl:value-of
select="CONTEXT/SUBCONTEXT[@TYPE='DYNAMIC']/SUBCONTEXTELEMENT[@NAME='SYSTEM']/SUBCONTEX
TELEMENT[@NAME='ICENTRICSERVICE']/SUBCONTEXTELEMENT[@NAME='REQUEST']/ancestor::SUBCONTE
XT/CATEGORY[@TYPE='LocationID']/@VALUE"/></xsl:attribute>
      </CATEGORY>
      <CATEGORY TYPE="DeviceID">
       <xsl:attribute name="VALUE"><xsl:value-of
select="CONTEXT/SUBCONTEXT[@TYPE='DYNAMIC']/SUBCONTEXTELEMENT[@NAME='SYSTEM']/SUBCONTEX
TELEMENT[@NAME='ICENTRICSERVICE']/SUBCONTEXTELEMENT[@NAME='REQUEST']/ancestor::SUBCONTE
XT/CATEGORY[@TYPE='DeviceID']/@VALUE"/></xsl:attribute>
      </CATEGORY>
      <SUBCONTEXTELEMENT NAME="SYSTEM">
       <SUBCONTEXTELEMENT NAME="ICENTRICSERVICE">
        <SUBCONTEXTELEMENT NAME="REQUEST"/>
       </SUBCONTEXTELEMENT>
      </SUBCONTEXTELEMENT>
     </SUBCONTEXT>
    </CONTEXT>
    <ONTOLOGY OID="#ICS-ontology: Device Request">
      The OID is just for testing purposes only!
     </ONTOLOGY>
    <XMLDATA/>
   </NOTIFY>
  </xsl:if>
 </xsl:template>
 <!-- ********************************************************
    *|@* - Template

    Description:  generic copy template
         copys all nodes and attributes to the
         output XML
```

```
 ********************************************************** -->
 <xsl:template match="*|@*" mode="copy">
  <xsl:copy>
   <xsl:apply-templates select="*|@*" mode="copy"/>
  </xsl:copy>
 </xsl:template>
</xsl:stylesheet>
```

7.5 SDO Platform Specific Model

This chapter presents a proposal for a CORBA-specific SDO model that was derived from the platform independent model described in chapter 4.5.2.

To derive the CORBA model, the interfaces and necessary data structures were mapped to OMG Interface Definition Language. The complete IDL definitions of types, structures, and interfaces can be found in the following. This IDL specification can be used to build SDO components that are compliant to the SDO model.

This CORBA model is based on the current CORBA 2.3 specification. That means, the CORBA component model (CCM) is not considered. However, because SDOs will have to implement several interfaces, the CCM is a promising candidate for later implementation. This would require deriving another PSM, which will be based on CCM. Such CCM-based model would then define a component IDL (CIDL) description of the data structures and interfaces.

The following sections describe the SDO CORBA model, starting with general data structures used by the interfaces, and then the interfaces themselves.

7.5.1 Data Structures

The following data structures, *DeviceProfile* and *SDOServiceProfile*, are derived from the resource data model and are used in methods of the *SDO* interface. Basic structures are not explained here in this chapter as detailed information is provided in the IDL file itself.

7.5.2 Structure Parameter

To define the *Parameter* structure, several additional data types are defined. A *Parameter* is defined by its name, the type of its value and optionally a data structure of type that defines limited value scope for this parameter. For string data types of parameters, the allowed values can be specified by an enumeration, in this case the field *allowedValues* contains structure *EnumerationType*. For numeric data types, the scope of original type can be limited by structures of type *RangeValue* or *IntervalValue*.

```
    enum NumericType {SHORT_TYPE, LONG_TYPE, FLOAT_TYPE,
   DOUBLE_TYPE};

    union Numeric switch (NumericType) {
        case SHORT_TYPE:  short    short_value;
        case LONG_TYPE:    long     long_value;
        case FLOAT_TYPE:   float     float_value;
        case DOUBLE_TYPE: double double_value;
    };

    struct EnumerationType {
        StringList enumeration_values;
    };

    struct RangeType {
        Numeric min;
        Numeric max;
        boolean min_inclusive;
        boolean max_inclusive;
    };

    struct IntervalType {
        Numeric min;
        Numeric max;
        boolean min_inclusive;
        boolean max_inclusive;
        Numeric step;
    };

    enum ComplexDataType {ENUMERATION, RANGE, INTERVAL};
```

```
    union AllowedValues switch (ComplexDataType) {
        case ENUMERATION: EnumerationType allowed_enum;
        case INTERVAL:        IntervalType        allowed_interval;
        case RANGE:            RangeType           allowed_range;
    };

    struct Parameter {
        string                    name;
        CORBA::TypeCode type;
        AllowedValues       allowed_values;
    };
```

7.5.2.1 Structure DeviceProfile

```
struct DeviceProfile {
 string  deviceType;
 string  deviceModel;
 string  manufacturer;
 NVList specifications;
};
```

7.5.2.2 Structure SDOServiceProfile

```
struct SDOServiceProfile {
 Identifier              serviceID;
 string                 interfaceReference;
 string                   serviceReference;
 string                   serviceDescription;
 Availability            availabilityStatus;
 ReservationStatus status;
     };
```

7.5.3 Interfaces

This section contains IDL descriptions for the *SDO* interfaces as introduced in the PIM. Each interface is mapped to a corresponding CORBA interface. The Interfaces defined in CORBA model are listed below:

- Interface SDO
- Interface Discovery
- Interface Monitoring
- Interface NotificationCallback
- Interface Configuration
- Interface ConfigurationExt
- Interface Reservation
- Interface SDOService

7.5.3.1 Exceptions

An exception in CORBA mapping contains attributes *errorCode* and reason. The exception codes for the SDO system are defined in enumeration *ExceptionCode*.

```
enum ExceptionCode {
     INTERFACE_NOT_IMPLEMENTED,
     SDO_NOT_FOUND,
     SERVICE_NOT_FOUND,
     ILLEGAL_ACCESS,
     INVALID_ARGUMENT,
     NOT_FOUND,
     PARAM_NOT_FOUND,
     OPERATION_FAILED,
     UNKNOWN
};

exception SDOException {
       ExceptionCode errorCode;
       string reason;
};
```

7.5.3.2 SDO Interface

Each SDO has to implement the general *SDO* interface. It provides access to the SDO attributes, such as SDO ID, and to the other interfaces, which are optional. The *SDO* interface inherits from the (extends) *Discovery* interface. This means SDO has to implement the operations defined in the *Discovery* interface, too.

```
interface SDO : Discovery {
 string                    getSDOID();
 string                    getSDOName();
 string                    getSDOType();
 NVList                    getSDOProperties();
 Availability          isAvailable();

              DeviceProfile          getDeviceProfile()
                                  raises(SDOException);
 SDOServiceProfileList   getServiceProfiles()
                                  raises(SDOException);
 SDOServiceProfile        getSDOServiceProfile(in Identifier serviceID)
         raises(SDOException);

 Monitoring           getMonitoring()
                                  raises(SDOException);
 Configuration           getConfiguration()
                                  raises(SDOException);
              ConfigurationExt          getConfigurationExt()
                                  raises(SDOException);
 Reservation           getReservation()
                                  raises(SDOException);
};
```

7.5.3.3 Discovery Interface

The PIM proposes to use broadcast communication mechanism, which is not sufficiently supported by CORBA products available today. In this PIM, the broadcast mechanism is not used but replaced with additional *announce()* operations in *Discovery* interface. So in order to publish presence announcements on joining an SDO system, an SDO has to get references of SDOs that are known and invoke their announce() operation.

Data Structures

The data structures used in *Discovery* interface are mapped straight forwardly from the PIM.

```
struct AnnouncementMsg {
         AnnouncementType msgType;
         SDO               sender;
         Identifer         senderID;
         string             sdoType;
         unsigned long duration;
         StringList        serviceDescriptionList;
};
struct SDOMask {
         Identifer sdoID;
         string      sdoName;
         string      sdoType;
         NVList     sdoProperties;
         boolean  available;
};

struct SDOServiceMask {
         Identifer              serviceID;
         NVList                 serviceProperties;
         string                  interfaceType;
         ReservationMode mode;
         boolean              available;
};

typedef sequence<SDOServiceMask> SDOServiceMaskList;
typedef sequence<SDOService>        SDOServiceList;
typedef sequence<SDO>                  SDOList;
```

Interface Definition

```
interface Discovery {

         void announce(in AnnouncementMsg announcement);
```

```
        SDOList  searchSDO (in SDOMask      sdoMask,
                            in DeviceProfile            deviceProfile,
                            in SDOServiceMaskList  serviceMaskList);

        SDOServiceList searchSDOService (in SDOMask  sdoMask,
                            in DeviceProfile  deviceProfile,
                            in SDOServiceMask serviceMask);
};
```

7.5.3.4 Interface Monitoring

The *Monitoring* interface provides on the one hand the direct interrogation of SDO state values (*getParametervalue()* method) and on the other hand it provides a subscription mechanism to these state values. Therefore, it defines a call-back interfaces, which has to be provided by the monitoring SDO and to be passed to the monitored SDO in the *NotificationSubscription* data structure (in the *subscribe()* method).

Data Structures

```
enum NotificationMode {ON_CHANGE, ON_INTERVAL};

struct NotificationSubscription {
 NotificationCallback subscriber;
 Identifer               subscriberID;
 StringList              observedData;
 NotificationMode     notifyMode;
 unsigned long         startTime;
 unsigned long         duration;
 unsigned long         notificationInterval;
};
```

Interface Definition

```
interface Monitoring {

        any getParameterValue (in string name)
                                  raises(SDOException);

        ParameterList   getMonitoringParameters ()
                                  raises(SDOException);

        NVList getCurrentStatus ()
                raises(SDOException);

        void subscribe (in NotificationSubscription data)
                              raises(SDOException);

        void renewSubscription (in Identifer     subscriber,
                                in unsigned long duration)
                          raises(SDOException);

        void unsubscribe (in Identifer  subscriber,
                          in StringList names)
                      raises(SDOException);

        void unsubscribeAll(in Identifer subscriber)
                      raises(SDOException);
};
```

Interface NotificationCallback

This is the call-back interface used for subscription. The monitoring SDO has to implement it and to pass its reference to the monitored SDO, which calls the *notify()* method according to the subscription (as specified in the *NotificationSubscription* data structure in the *subscribe()* method).

```
interface NotificationCallback {
        void  notify (in Identifer senderID,
                      in NVList currentStatus)
                raises(SDOException);
};
```

7.5.3.5 Interfaces Configuration and ConfigurationExt

The interface *Configuration* contains basic operations required for configuration. It used to modify the values of SDO and service parameters.

Data Structures

For the interface *ConfigurationExt*, the data structure for stored configurations, *StoredConfiguration*, is defined.

```
struct StoredConfiguration {
 Identifer  configID;
 string      comment;
 NVList     configurationData;
    };

typedef sequence<StoredConfiguration> StoredConfigurationList;
```

Interface Definition

```
interface Configuration {
 ParameterList getConfigParameters () raises(SDOException);
 any getParameterValue(in string name) raises(SDOException);
 void modifyConfigParameter (in Identifier sdoID,
   in string name,
   in any   value)
   raises(SDOException);
 NVList getConfiguration () raises(SDOException);
};
```

The interface *ConfigurationExt* contains operations for management of stored configurations. The *ConfigurationExt* interface extends the *Configuration* interface. The SDO has to implement the *Configuration* or the *ConfigurationExt* interface.

Clients can always access the current configuration via the *Configuration* interface, even if the *ConfigurationExt* interface is implemented.

```
interface ConfigurationExt : Configuration{
 StoredConfigurationList    getStoredConfigList() raises(SDOException);
 void activateConfig (in Identifier sdoID,
   in Identifier configID) raises(SDOException);
 void addNewConfig(in StoredConfiguration configuration)
   raises(SDOException);
 void modifyConfig(in Identifier configID,
   in NVList     values)
   raises(SDOException);
 void removeConfig(in Identifier configID) raises(SDOException);
};
```

7.5.3.6 Interface Reservation

Data Structures

```
enum ReservationMode   {FREE, RESERVED, EXCLUSIVE_RESERVED};
enum ReservationResult  {OK, NOK, NOT_AVAILABLE};

struct ReservationStatus {
      ReservationMode   mode;
      unsigned long       availableAfter;
    };

struct ReservationResponse {
      ReservationResult result;
      unsigned long         availableAfter;
    };
```

Interface Definition

```
interface Reservation {
 ReservationStatus getReservationStatus (in Identifier resourceID)
   raises(SDOException);
 ReservationResponse reserve(in Identifier      resourceID,
   in unsigned long      duration,
   in ReservationType  type)
   raises(SDOException);
 void cancelReservation(in Identifier sdoID,
```

```
   in Identifier       resourceID)
   raises(SDOException);
};
```

7.5.3.7 Interface SDOService

The *SDOService* interface is the base interface for SDO services. SDO service interfaces are SDO specific and not standardized. However, each SDO service has to implement the *SDOService* interface (using inheritance mechanisms) in order to support its configuration.

```
interface SDOService {
 Configuration getConfiguration() raises(SDOException);
};
```

7.5.4 The complete IDL

```
// SDOPackage.idl

#ifndef    _SDO_PACKAGE_IDL_
#define    _SDO_PACKAGE_IDL_

#include <corba.idl>
#include <CosNotifyComm.idl>

/**
 * CORBA specific model for SDOs
 */

#pragma prefix "org.omg"
#define exception_body { string description; }

module SDOPackage {
    interface SDO;
    interface SDOService;
    interface SDOSystemElement;
    interface Configuration;
    interface Monitoring;
    interface Organization;

    /** ------- Data Types -------*/
    typedef sequence<string> StringList;
    typedef sequence<SDO> SDOList;
    typedef sequence<Organization> OrganizationList;
    typedef string UniqueIdentifier;

    struct NameValue {
 string name;
 any value;
    };

    typedef sequence<NameValue>   NVList;

    struct OrganizationProperty {
 NVList properties;
    };

    enum DependencyType {
 NORMAL,
 REVERSE,
 NO_DEPENDENCY
    };

   enum NumericType {
    SHORT_TYPE,
    LONG_TYPE,
    FLOAT_TYPE,
    DOUBLE_TYPE};

    union Numeric switch (NumericType) {
 case SHORT_TYPE:  short short_value;
 case LONG_TYPE: long long_value;
 case FLOAT_TYPE: float float_value;
 case DOUBLE_TYPE: double double_value;
    };

    struct EnumerationType {
 StringList enumerated_values;
    };

    struct RangeType {
 Numeric min;
```

```
Numeric max;
boolean min_inclusive;
boolean max_inclusive;
   };

   struct IntervalType {
Numeric min;
Numeric max;
boolean min_inclusive;
boolean max_inclusive;
Numeric step;
   };

   enum ComplexDataType {
    ENUMERATION,
    RANGE,
    INTERVAL};

   union AllowedValues switch (ComplexDataType) {
case ENUMERATION:   EnumerationType allowed_enum;
case INTERVAL:      IntervalType    allowed_interval;
case RANGE:         RangeType       allowed_range;
   };

   struct Parameter {
string         name;
CORBA::TCKind  type;
AllowedValues allowed_values;
   };

   typedef sequence<Parameter> ParameterList;

   struct DeviceProfile {
string device_type;
string manufacturer;
string model;
string version;
NVList properties;
   };

   struct ServiceProfile {
string id;
string interface_type;
NVList properties;
SDOService service;
   };

   typedef sequence<ServiceProfile> ServiceProfileList;

   struct ConfigurationSet {
string id;
string description;
NVList configuration_data;
   };

   typedef sequence<ConfigurationSet> ConfigurationSetList;

   /** ------- Exceptions -------*/

   exception NotAvailable            exception_body;
   exception InterfaceNotImplemented exception_body;
   exception InvalidParameter        exception_body;
   exception NotFound                exception_body;

   /** ------- Interfaces -------*/
   interface SDOSystemElement {
OrganizationList get_organizations()
 raises (NotAvailable);
   };

   interface SDO : SDOSystemElement {
UniqueIdentifier get_id()
 raises (NotAvailable);
string get_SDO_type()
 raises (NotAvailable);
DeviceProfile get_device_profile ()
 raises (NotAvailable);
ServiceProfileList get_service_profiles ()
 raises (NotAvailable);
ServiceProfile get_service_profile (
 in string id
 ) raises (InvalidParameter, NotAvailable);
SDOService get_service (
 in string id
 ) raises (InvalidParameter, NotAvailable);
Configuration get_configuration ()
    raises (InterfaceNotImplemented, NotAvailable);
```

```
 Monitoring get_monitoring ()
     raises (InterfaceNotImplemented, NotAvailable);
    };

    interface Configuration {

 void set_device_profile (in DeviceProfile dProfile)
  raises (InvalidParameter, NotAvailable);
 void add_service_profile (in ServiceProfile sProfile)
  raises (InvalidParameter, NotAvailable);
 void add_organization (in Organization organization)
  raises (InvalidParameter, NotAvailable);
 void remove_device_profile ()
  raises (NotFound, NotAvailable);
 void remove_service_profile (in string id)
  raises (InvalidParameter, NotAvailable);
 void remove_organization (in Organization organization)
  raises (InvalidParameter, NotAvailable);
 ParameterList get_config_parameters ()
  raises (NotAvailable);
 any get_parameter_value (in string name)
  raises (InvalidParameter, NotAvailable);
 void set_config_parameter (in string name,
       in any value)
  raises (InvalidParameter, NotAvailable);
 ConfigurationSetList get_configuration_sets ()
  raises (NotAvailable);
 void add_configuration_set (in ConfigurationSet configuration_set)
  raises (InvalidParameter, NotAvailable);
 void remove_configuration_set (in string config_id)
  raises (InvalidParameter, NotAvailable);
 void activate_configuration_set (in string config_id)
  raises (InvalidParameter, NotAvailable);
    };

    interface Monitoring : CosNotifyComm::StructuredPushConsumer,
    CosNotifyComm::StructuredPushSupplier {
 any get_parameter_value (in string name)
  raises (InvalidParameter, NotAvailable);
 ParameterList get_monitoring_parameters ()
  raises (NotAvailable);
 NVList get_current_monitoring_status ()
  raises (NotFound, NotAvailable);
    };
    interface SDOService {};

    interface Organization {
 void add_organization_property (
  in OrganizationProperty organization_property
  ) raises (InvalidParameter, NotAvailable);
 void remove_organization_property (
  in string name
  ) raises (NotFound, NotAvailable);
 OrganizationProperty get_organization_properties ()
  raises (NotAvailable);
 SDOList get_members ()
  raises (NotAvailable);
 void set_members (
  in SDOList sdos
  ) raises (InvalidParameter, NotAvailable);
 SDOSystemElement get_owner ()
  raises (NotFound, NotAvailable);
 void set_owner (
  in SDOSystemElement sdo
  ) raises (InvalidParameter, NotAvailable);
 DependencyType get_direction()
  raises (NotAvailable);
 void set_direction (
  in DependencyType direction
  ) raises (NotAvailable);
    };
};
#endif //_SDO_PACKAGE_IDL_
```

7.6 List of Figures and Tables

7.6.1 List of Figures

7.6.2 List of Tables