Orchestrating Secure Workflows for Cloud and Grid Services

vorgelegt von
Dipl.-Inform.
André Höing
aus Berlin

Von der Falultät IV - Elektrotechnik und Informatik
der Technischen Universität Berlin
zur Erlangung des akademischen Grades
Doktor der Ingenieurwissenschaften
- Dr. Ing. -

genehmigte Dissertation

Promotionaausschuss:

Vorsitzender:	Prof. Dr. Hans-Ulrich Heiß
Gutachter:	Prof. Dr. Odej Kao
	Prof. Dr. Wilhelm Hasselbring

Tag der wissenschaftlichen Aussprache: 07.10.2010

Berlin 2010

D83

Acknowledgement

I would like to show my gratitude to all the people who accompanied me during my studies and my PhD and made this thesis possible. I especially want to thank my advisor Odej Kao for his guidance and inviting me to work in his group. My appreciation also go to Wilhelm Hasselbring for agreeing to review this thesis.

This work would not have been possible without my colleagues Guido Scherp and Stefan Gudenkauf from Oldenburg. Thanks for all the online meetings, discussions, and the great teamwork during the BIS-Grid project with all its ups and downs. I would also like to mention all other project partners and thank them for the successful cooperation.

Special thanks also go to Matthias Hovestadt, Philipp Berndt, Daniel Warneke, Dominic Battré, Guido Scherp, and Martin Raack for proofreading my thesis.

Thanks go also to the entire working group “Complex and Distributed IT Systems” at TU-Berlin for the awesome past three years. Thanks for making this job probably the best in my life. You are more than colleagues for me. We had a great time and I hope to stay in contact with you forever. I also thank my former colleague and friend Felix Heine for encouraging me to start my PhD after finishing my Diploma thesis.

Finally, I would like to thank my family for their support during all the time. A special thanks goes to my dear girlfriend Ellen for her love and patience especially during writing this thesis.

Abstract

The modern design of business and scientific IT landscapes is based upon service-oriented architectures. In doing so, small functional units are encapsulated as services accessible via standardized interfaces. Grid and Cloud computing both employ such services: Grid computing provides access to mostly scientifically used compute resources and data. Cloud computing is a commercially-driven, recently emerging technology that offers services to access compute power, platforms, and software solutions. Nowadays, workflows are the preferred means for the combination of services into added value service chains representing functional business processes or complex scientific experiments. However, the advantages of integrating services from all three domains (business, science, and Cloud computing) into one workflow is not exploited sufficiently. The integration of external services within workflows raises various challenges.

This thesis presents a novel architecture for a workflow engine that is capable of integrating services from all three domains. It starts with an analysis of workflow life cycles and infers requirements from these. Additionally, it considers the idea of providing a Cloud service for workflow orchestration. This service offers effective means to deploy and execute workflows completely on external resources. Simultaneously, it allows elasticity and fair billing models in the context of Cloud computing. The resulting architecture is based on standards without creating a new proprietary workflow description language dialect in order to increase acceptance and sustainability. It especially targets security and communication barriers, which originate from the diversity of service providers as well as handling stateful services with default WS-BPEL activities.

The architecture is evaluated with respect to its requirements and in two exemplary real-life business scenarios. Furthermore, the thesis illustrates the general applicability, with some limitations, to scientific workflows with the help of example workflows for typical scientific tasks. Finally, it analyses the performance of the new components.

Zusammenfassung

Modernes Design betrieblicher und wissenschaftlicher IT Landschaften basiert auf service-orientierten Architekturen. Dabei werden kleine funktionale Bestandteile durch Dienste mit standardisierten Schnittstellen gekapselt. Sowohl Grid als auch Cloud Computing benutzen solchen Dienste: Grid Computing bietet Zugang zu meist wissenschaftlich genutzten Rechen-Resourcen und Daten. Cloud Computing ist eine kommerziell getriebene und an Bedeutung zunehmende neue Technologie, die Dienste für den Zugriff auf Rechenleistung, Plattformen und Software-Lösungen bereit stellt. Heutzutage sind Workflows das bevorzugte Mittel zur Kombination von Diensten zu Wertschöpfungsketten, die betriebliche Prozesse oder komplexe wissenschaftliche Experimente repräsentieren. Dennoch werden die Vorteile einer Integration von Diensten aus allen drei Bereichen (betriebliches, wissenschaftliches und Cloud Computing) in einen Workflow nicht ausreichend ausgeschöpft. Diese Integration von externen Diensten in Workflows wirft verschiedenste Fragestellungen auf, z.B. im Bezug auf Kompatibilität und Sicherheit.

Die Arbeit präsentiert eine neuartige Architektur für eine Workflow Engine, welche die Integration von Diensten aus allen drei Bereichen ermöglicht. Sie beginnt mit der Analyse von Workflow Lebenszyklen und leitet hieraus Anforderungen ab. Zusätzlich wird noch die Idee eines Cloud-Dienstes zur Dienstorchestrierung betrachtet. Solch ein Dienst bietet effektive Mittel zum Einrichten und Ausführen von Workflows auf externen Ressourcen. Gleichzeitig ermöglicht er Elastizität und faire Abrechnungsmodelle im Kontext von Cloud Computing. Die sich daraus ergebende Architektur basiert auf Standards ohne einen neuen proprietären Workflow-Dialekt zu erzeugen, um die Akzeptanz und Nachhaltigkeit zu erhöhen. Dabei werden besonders Sicherheits- und Kommunikationsbarrieren behandelt, die sich aus der Dienstanbietervielfalt und dem Umgang mit zustandesbehafteten Diensten ergeben.

Die Architektur wird in Bezug auf die erarbeiteten Anforderungen und in zwei exemplarischen, realen betrieblichen Szenarios evaluiert. Weiterhin veranschaulicht die Arbeit die generelle Anwendbarkeit, mit geringen Einschränkungen, für wissenschaftliche Workflows anhand von Beispielen für typische wissenschaftliche Aufgaben. Schließlich wird die Performanz der neuen Komponenten analysiert.

Contents

1 Introduction

Contents

The cooperation of departments, enterprises and scientific institutes is an important factor for successful work. Such cooperations yield in significant synergy effects since selected information systems may be outsourced to partners. With regard to business information technologies (IT), outsourcing means purchasing services from partners and therewith reducing costs for IT services since the partner "can achieve economies of scale, economies of scope, and economies of specialization" [11].

Modern technologies such as *service-oriented architectures* (SOA) [97] allow the forming of different IT systems to a coherent IT landscape. Such architectures enable the mapping of functional business processes on a technical system level. Small business functions are encapsulated in loosely-coupled services which are combined to executable workflows. This allows a flexible integration of enterprise IT services that are hosted in-house. Integrating outsourced IT or systems hosted at specialized service providers entails manifold challenges for both partners such as security.

Workflow management is the technology of choice for business processes, realizing a well-defined and orchestrated execution of multiple inter-dependent tasks. Numerous commercial workflow management systems are available, like Microsoft BizTalk, Oracle Business Process Manager, and Sun's Java Composite Application Platform Suite which supports companies in the technical realization of workflows. Furthermore, several open-source products underline the large market for service composition. While the workflow management systems mentioned above differ significantly in their architecture and functionality profile, WS-BPEL [75] has evolved as de-facto standard for workflow description. Such mostly expensive systems are already established in big enterprises, the introduction of service-oriented architectures and professional workflow management tools proceeds only slowly in small companies [21]. However, the direct technical support of processes is an important factor for the effectiveness of all companies.

The term eScience abstracts the execution of complex scientific experiments as simulations executed on compute resources. Today, such experiments are of enormous importance in numerous scientific domains such as physics, astronomy, informatics, and health. The realization demands the cooperation of different scientific domains and institutes to exploit the full potential of large data centers. This demand constitutes Grid computing [49, 80, 81] that has evolved as a well established instrument for sharing compute and data resources. Scientists are provided with access to external resources allowing them to execute complex experiments and simulations in a short period of time compared to the duration that the experiment would have taken on their office computers or smaller local clusters.

The scientific domain also uses workflows for the realization of simulated experiments that facilitate the cooperation of different sites and data centers. Scientific workflows are usually mapped to Grid services. Such services are stateful and high secured Web services providing access to high performance compute resources. Grid middleware systems provide mechanisms for executing stateful Grid services complying to industry level standard like WSRF [103]. These middleware systems are following general SOA principles since they encapsulate typical high performance compute functions as small services. In contrast to the business domain, the scientific community did not agreed on a standard for workflow description. Hence, many scientific communities created their individual workflow description languages, tailored to specific requirements and available execution engines, but incompatible to other available languages.

Grid computing shared usage of large resources results in cost-efficiency since purchasing a large resource for only one experiment would not be sustainable. Although cost-efficiency is from enormous importance even for the business domain, Grid computing has not been accepted as default technology there. This possibly originates from partly missing features like quality of service agreements and the extreme focus on high performance compute jobs instead of general information system provisioning. Thus, the IT market leaders strive to develop new means for providing all kind of IT services to business customers.

Cloud computing is the new paradigm for externally hosted IT in a commercial way. It exactly targets the trend for cooperation and outsourcing with the provisioning of infrastructure, platforms, and software in a cheap and scalable way to a broad range of customers. Cloud providers usually operate huge data centers to exploit the advantages of the economies of scale and to guarantee a well-defined degree of quality of service. Buyya et al. in [25] and Armbrust et al. in [10] anticipate that we will see a plethora of Cloud computing offerings such as computers, data, and instruments in the near future. This will force the evolution from an already existing InterGrid – the cooperation between different scientific institutes – to a new InterCloud – a (loose) partnership between providers and users. Cloud services often are cheaper than services on dedicated possibly outsourced

resources since modern technologies enable the concurrent usage of hardware resources without the risk of interleaving customers. This is denoted as the multi-tenant architecture of Cloud systems.

Cloud providers use proprietary Web services frameworks enriched with various technologies (e.g. for security) for the realization of the actual services. Hence, there is no common standard which would enable interoperability between Cloud computing products. Additionally, the combination of cross-provider Cloud services to workflows is not possible.

1.1 Problem Definition

As described in the introduction, service-oriented architectures has evolved as a mature technology, established in the academic and commercial domain. It facilitates the integration of different applications by means of common interface descriptions and communication protocols. Additionally, the need for cooperation increases constantly due to an increasing complexity of business and scientific tasks. The increasing acceptance of Grid and Cloud technologies in all domains underlines the need for external resources. However, the technical realization of such integration scenarios even across administrative boundaries is still difficult because of incompatible service frameworks.

Figure 1.1 depicts a small and exemplary cutout of typical IT landscapes, all based on SOA. It outlines the separated domains of business and science as well as the newly upcoming Cloud services. The exemplary data centers are annotated with information according commercial information systems, Grid middlewares, and Cloud providers to underline the diversity of used Web service frameworks and middlewares. In particular, the IT of small and medium enterprises is often still entirely separated from the outside world for security reasons. At the same time, some of them apply a service-oriented approach for realizing internal IT and workflows for mapping the business processes on this landscape.

In contrast to this, the scientific domain already cooperates in cross-institutional communities. Such communities usually agree to use a single Grid middleware system as well as a workflow engine (e.g. [109, 38, 104]) supporting this Grid middleware system, offering all required functions. Furthermore, communities often develop workflow engines tailored to their particular requirements. Thus, some scientific workflow engines are based on older and partly non SOA-enabled middleware releases that are incompatible to modern Grids. For improving the level of interoperability, some workflow engines (e.g. Triana [119]) are using an additional abstraction layer allowing the mapping of the workflow on different middlewares.

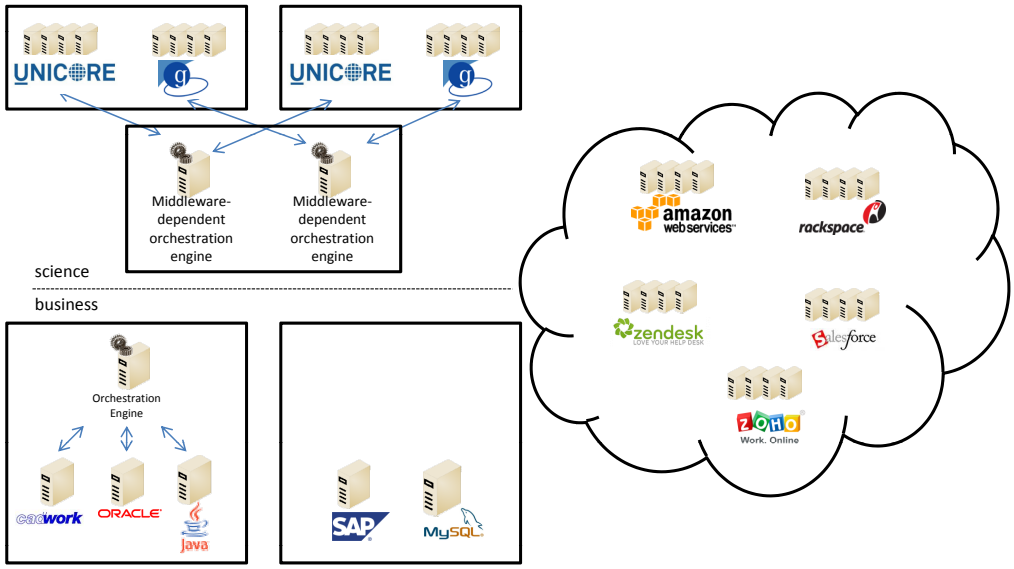


FIGURE 1.1: Service landscapes including business, scientific, and Cloud systems

On the right hand side, Figure 1.1 depicts some commercial Cloud providers offering diverse services to end-users. Each Cloud provider offers a portfolio of services on different levels of abstraction such as access to compute power and data, frameworks for hosting customer-developed software, and comprehensive software stacks in the sense of application service provisioning.

This thesis targets the realization of business processes and scientific workflows that combines various services from all of the above mentioned service types. This will increase interoperability between the different domains as well as between the services inside one domain and ease switching between service providers. In doing so, we consider stateless as well as stateful services according to the WSRF [103] standards. For workflow orchestration, we have chosen the WS-BPEL standard as it is well established and tested in this business domain since several years. The capability to integrate services from different providers will alleviate the cooperation of enterprises and scientific institutes significantly and help them to concentrate on the respective core competencies instead of being hampered by proprietary IT and interoperability details.

The central component of the architecture is a workflow orchestrator engine that serves as interoperability interface between the different services types since it supports different technologies and standards to overcome the technical barriers. Additionally, customers will benefit from the architecture since it is also applicable to be offered as an orchestra-

tion Cloud service in a shared environment. This allows for billing workflow execution with fair pricing models, for example on an on-demand and pay-per-usage basis. These properties will increase the attractiveness of IT outsourcing and enterprise application integration for small and medium enterprises which would not be able to profit from the advantages of SOA, otherwise.

Also scientists will benefit from additional flexibility and the possibility to integrate different service types in one workflow since it will allow a faster realization of experiments with services that do not fit in the existing Grid middlewares. For instance, self-written and hosted services can be used instead of implementing more complex Grid services that additionally have to be deployed by the partner on his hardware.

The main challenges when designing the architecture can be summed up in the following sub-goals:

1. Overcoming technical barriers to enable the integration of different service types in one workflow.
2. Handling stateful and stateless service implementations efficiently in one workflow.
3. Encapsulating workflows again inside services to allow hierarchical workflow compositions.
4. Realizing a fine-grained and role-based access control system that is applicable in the scientific domain and particularly the commercial domain as well.
5. Designing the orchestration service as a multi-tenant architecture to enable a cheap and efficient workflow provisioning.

1.2 Contribution

This thesis describes the requirements, design, implementation, and evaluation of a new orchestration architecture that facilitates the integration of stateful and stateless services across organizational boundaries. It considers requirements from the business but as well from the scientific domain to enable a modern form of (enterprise) application integration. The architecture provides solutions to the following three challenges:

Firstly, it allows even small and medium enterprises to employ outsourcing as a means to lower costs. In the context of Cloud computing, several commercial tools for standard business tasks like customer relationship or accounting and billing are available as comprehensive service products on the Internet. The architecture empowers small and medium enterprises to integrate Cloud services with in-house or other externally hosted IT services.

Secondly, the architecture describes a flexible workflow engine for eScience that is not limited to Grid middlewares but is extendable to integrate new services, e.g. commercial Cloud services or self-hosted more simple Web services. Thus, scientists become more independent of the complex Grid technologies and get the opportunity to self-provide services or to combine other services with Grid computing. This will speedup science since it allows a faster and more flexible service usage especially interesting for scientific communities whose focus is another than computer science and therefore have difficulties with setting up and maintaining Grid middlewares. Since the actual workflow orchestration is based on WS-BPEL, the scientific domain will possibly profit from further advances in the business domain instead of creating further proprietary workflow engines.

Thirdly, the architecture allows customers (and scientists) to deploy and run their workflows on an external infrastructure. Such a service must be designed as a multi-tenant architecture to facilitate high hardware utilization and therewith a cheap and fair pricing model. Since the operation of a workflow engine requires a lot of know-how and money, it exceeds the capabilities of many small enterprises. Often such enterprises only require few workflows so that already the license costs of a professional workflow engine would exceed the budgets. Thus, a Cloud service for workflow execution will enable such enterprises to integrate own and partners' IT systems with adequate costs.

Parts of this thesis have been released in the following publications:

1. **S. Gudenkauf, W. Hasselbring, F.Heine, A. Höing, O. Kao, G. Scherp**
   BIS-Grid: Business Workflows for the Grid
   In: Proceedings of Cracow Grid Workshop 2007 (CGW07), ACC CYFRONET AGH, 2008, pp. 86-93

2. **S. Gudenkauf, W. Hasselbring, F. Heine, A. Höing, O. Kao, G. Scherp**
   A Software Architecture for Grid Utilisation in Business Workflows
   In: Proceedings of Multikonferenz Wirtschaftsinformatik (MKWI08), GTO-Verlag Berlin, 2008, pp. 91-102

3. **S. Gudenkauf, W. Hasselbring, A. Höing, G. Scherp, O. Kao**
   Workflow Service Extensions for UNICORE 6 – Utilizing a Standard WS-BPEL Engine for Grid Service Orchestration
   In: Proceedings of Euro-Par 2008 Workshops - Parallel Processing (UNICORE08), Springer, 2009, pp. 103-112

4. **A. Brinkmann, S. Gudenkauf, W. Hasselbring, A. Höing, O. Kao, H. Karl, H. Nitsche, G. Scherp**
   Employing WS-BPEL Design Patterns for Grid Service Orchestration using a Standard WS-BPEL Enngine and a Grid Middleware
   In: Proceedings of Cracow Grid Workshop 2008 (CGW08), ACC CYFRONET AGH, 2009, pp. 103-110

5. **A. Höing, G. Scherp, S. Gudenkauf, D. Meister, A. Brinkmann**
   An Orchestration as a Service Infrastructure using Grid Technologies and WS-BPEL
   In: Proceedings of the 7th International Joint Conference on Service-Oriented Computing (ICSOC-ServiceWave '09), Springer, 2009, pp. 301-315
6. **A. Höing, G. Scherp, S. Gudenkauf**
   The BIS-Grid Engine: An Orchestration as a Service Infrastructure
   International Journal of Computing, Volume 8, 2009, pp. 96-104
7. **S. Gudenkauf, G. Scherp, W. Hasselbrink, A. Höing, O. Kao**
   Workflow Modeling for WS-BPEL-based Service Orchestration in SMEs
   In: Proceedings of the Software Engineering 2010 Workshops
   Grid Workflow Workshop (GWW10), GI, 2010, pp. 185-192
8. **S. Gudenkauf, W. Hasselbring, A. Höing, G. Scherp, O. Kao**
   Using UNICORE and WS-BPEL for Scientific Workflow Execution in Grid Environments
   In: Proceedings of Euro-Par 2009 Workshops - Parallel Processing (UNICORE09), Springer, 2010, pp. 335-344

1.3 Outline of the Thesis

The remainder of this thesis is structured as follows:

Chapter 2: Fundamentals

Chapter 2 gives background information that helps to understand the design decisions and finally the architecture. The section includes short introductions into the following topics: service-oriented architectures, the workflow execution in SOAs, an overview of Grid computing and Grid security as well as an introduction into the Cloud computing.

Chapter 3: Requirements for secure Workflow Orchestration

Chapter 3 presents and compares the two main application scenarios covered by the thesis: business and scientific workflows. Additionally, we concrete the idea of offering a workflow engine as Cloud service to allow cheap and reliable orchestration without running an in-house workflow engine. Based on these scenarios, we conclude with a list of requirements for the targeted workflow orchestration architecture.

Chapter 4: Orchestration Architecture

Chapter 4 constitutes the main chapter of the thesis. It presents the general architecture of the workflow engine, denoted as Grid and Cloud workflow engine, including a component overview, the security infrastructure, and the plug-in mechanism for supporting various external service providers. Some concept are explained by means of the prototype implementation, the BIS-Grid workflow engine. Furthermore, the chapter briefly discusses some ideas how human interaction can be realized with respect to the cooperation scenarios. It concludes with a review of the architecture considering the requirements derived in Section 3.3.

Chapter 5: Prototype

Chapter 5 briefly describes the prototype implementation and highlights the realization of the most important concepts. Additionally, it presents the BIS-Grid deployment package which includes all information on workflow deployment.

Chapter 6: Evaluation

Chapter 6 presents the evaluation of the BIS-Grid workflow engine. We examined the applicability to business scenarios in two commercial application scenarios that were part of the BIS-Grid project. Additionally, we illustrate the feasibility of the engine with respect to the integration of Grid services for scientific workflows and the integration of Cloud services. Lastly, we debate a short performance evaluation.

Chapter 7: Related Work

Chapter 7 discusses related work. We mainly focus on papers about Grid orchestration, first approaches on Cloud workflows, and the adaption of Grid workflow engines to Cloud services. Related work on Web service orchestration is only outlined shortly because the orchestration of Web services has become a mainstream technology today.

Chapter 8: Conclusion

Chapter 8.2 concludes the thesis with a summary and an outlook on future work.

2 Fundamentals

Contents

This chapter describes the fundamental basic technologies that are used for Enterprise Application Integration (EAI) today. Nowadays, EAI is realized through service-oriented architectures often realized with Web service technologies. Additionally, Cloud and Grid computing are modern technologies in the scientific and business information technology domain. Both are also based on the SOA paradigm.

Firstly, we provide some background about service-oriented architectures. Therein, business functions are encapsulated in loosely-coupled services, mostly based on Web service technologies. Secondly, we introduce workflows, which are a means to combine such services in a structured way. The industrial de-facto standard for workflows in SOA is the *Web Service Business Process Execution Language* (WS-BPEL). WS-BPEL offers the possibility to map business processes to the technical system level by describing the processes – as far as possible – in an executable workflow description language. Workflow technologies are widely accepted in both, the science and the business domain, but require high setup and maintenance costs. Thus, often only big enterprises and scientific data centers have the ability to run a comprehensive SOA.

By extending the standard EAI solutions and offering possibilities to connect modern computing infrastructures, these technologies will become more attractive. Grid and Cloud computing are such modern computing infrastructures offering a cheap and very flexible means to access large data centers, software applications, or comprehensive IT solutions. The last two sections of this chapter outline the main characteristics for Grid and Cloud computing.

2.1 Service-oriented architectures – SOA

Nowadays, the architecture of modern IT landscapes is an important but also very complex topic in enterprises. The term *service-oriented architecture* is a buzzword for the design of such landscapes since it was formed in the end of the nineties. Since then, the term was used in various contexts and without a unique definition. An OASIS Technical Committee tried to develop a document that describes the most important aspects of a SOA as a reference model [97]. However, this reference model only defines some concepts, principals, and relationships that are necessary for the design of a SOA. A concrete architecture or implementation is beyond the scope of the reference model.

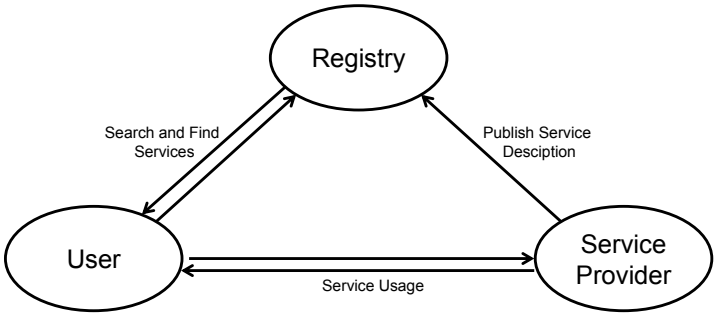


FIGURE 2.1: SOA triangle - main components of an service-oriented architecture

Services are the main concept of a SOA. A service refers to an arbitrary piece of software that offers a well-defined function via standardized interfaces. A service can be used by other services, legacy applications, or customers. This requires that service interfaces and communication protocols must be platform neutral. SOA services are loosely-coupled and can be reused whenever their functionality is required. A services based IT architecture requires a registry as a central point of information where all available services are listed. This list includes the general interfaces of each service but also technical details to invoke it. Hence, a SOA generally consists of three components: a registry, service providers

and their services, and the service users. As depicted in Figure 2.1, the service provider pushes the descriptions of his services into the registry. This again offers means to search for services using various criteria. As result of such an query, the user gets a list of appropriate services and the technical information to invoke them.

When introducing SOA to an IT landscape, there is not necessarily a direct mapping of services to existing applications. Services should be designed according to tasks in business processes and form small and reusable units. Therefore, a service possibly needs to access different databases or backend applications. This is a paradigm shift from solution islands, where one application exactly solves one problem, towards a layer of loosely-coupled services (cf. [78]).

The SOA reference model makes no assumption about the technologies that should be used for building a SOA. Generally, nearly all programming languages or technologies for distributed systems can be used, because on a very high level of abstraction only the cross-platform communication is a requisite. However, if the SOA should be interoperable, it is necessary that the communication is also independent from the service implementation. For example, Java RMI can be used to create a SOA but integration with other programming languages is difficult. Today, Web services are the de-facto standard used for the realization of SOAs. Such services use SOAP [60] and Internet protocols like HTTP for message exchange. Service descriptions are published using the Web Services Description Language (WSDL) [30]. All these technologies are platform independent and enjoy significant support from the industry. Today, several proprietary and open-source frameworks are available as well as tools for Web service implementation and orchestration of Web services with WSDL and SOAP.

Beside SOAP-based Web services, also other communication protocols are used in modern SOAs, depending on the complexity. REST (Representational State Transfer) [45] services have become more and more popular during the last years because the protocol partly works without XML. REST communication is closely related to the HTTP protocol, as it supports the following HTTP operations on stateful Web services. A resource is an object that represents the state of the service.

- GET: Requests the resource representation. The operation has no effect on the resource state.
- POST: Adds or modifies a resource. For example, this can cause an update in a database.
- PUT: Creates new resources or replaces a resource.
- DELETE: Deletes a resource.

In contrast to SOAP-based services, a RESTful service does not publish its interface in a standardized way. Instead, setting up a communication requires detailed knowledge about the interfaces. In compliance with the HTTP standard, parameters are submitted via the URL or as HTTP content – normally as XML. The advantage of REST services is its simplicity. It guarantees a high scalability because of a small software stack. However, the missing possibility for interface descriptions and the propagation of errors as HTTP error codes hamper a debugging of the communication compared to SOAP.

The loosely-coupling of Web services offers various options for service combination for example as executable workflows. The most used techniques for workflow implementation are *service orchestration* and *service choreography*. While service orchestration is the arrangement of services from a central instance, service choreography routes messages in a peer-to-peer style directly between the participating services. Both ways have advantages and disadvantages, e.g. regarding communication overhead, monitoring, or flexibility. In industrial scenarios, like the integration of information systems, service orchestration is preferred as a direct mapping of business processes to technical workflow descriptions is possible. Furthermore, a central orchestrator can easily offer capabilities for business process monitoring. Section 2.2 will introduce the Web Services Business Process Execution Language (WS-BPEL) in more detail and compare some implementations of WS-BPEL-based orchestrator services.

Since SOAP has a much longer history and SOAP-based Web services provide their interfaces in a well-defined way, several workflow engines for SOAP based communication are available on today's market and well established in the business domain. Workflow engines for RESTful services are rare, except some examples like [35]. The development of a comprehensive and accepted workflow language that supports RESTful services is still open. As a workaround, IBM presented a tutorial on how to map RESTful interfaces to WSDL interfaces in [96]. It highlights the possibility to integrate RESTful services in SOAP based workflow engines if a mapping between the WSDL interface and the REST calls is available.

To conclude, a SOA is a paradigm to integrate different information systems and to interconnect legacy systems. All components are implemented as or encapsulated by loosely-coupled services. For this, different technologies can be used as long as they all support at least a common communication protocol for message exchange. Through orchestration, informal business processes can be mapped to the technical system level. For example, workflows help to develop a consistent user interface that integrate several backend applications as a single service. The complexity of the involved interfaces in the backend communication is hidden transparently from the user client. Without SOA, the user would have to handle different application clients, each communicating with its own backend counterpart. Such simplified user clients will increase the users' work speed and lower training costs.

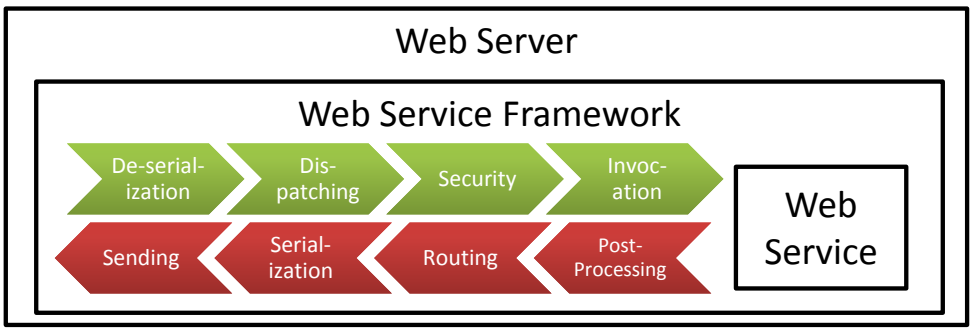


FIGURE 2.2: Typical architecture of an Web service framwork

2.1.1 SOAP Message Processing

Many different open source Web service frameworks are available (e.g. Apache Axis, Apache CXF, XFire, or ASP.NET). Such frameworks offer means to implement Web services and to handle SOAP messages in a structured way. The general functionality is the same for all of these implementations. The SOAP message (incoming or outgoing SOAP message) is processed by a so-called handler pipeline. Each handler in such a pipeline fulfills a small but well-defined functionality during message processing. This typical message processing architecture is depicted in Figure 2.2 as an example. The incoming pipeline firstly de-serializes the message. The second handler uses the data to find the target service while the third handler checks the message for included security tokens and processes access control. The actual invocation on the service implementation is done by the fourth handler. The outgoing message pipeline includes further handlers for example for post-processing, evaluation of routing information, message serialization, and message sending.

Handlers possibly require a well defined execution order because some handlers depend on information from previously executed handlers. To pass this information through the handler pipeline, there is often a common data structure that is passed through the whole message processing pipeline. We call this object the *message context*. It comprises all message and handler dependent information as security tokens, the actual message(s), or arbitrary other information that is stored by one handler and reprocessed by another one.

In most cases, the frameworks offer the possibility to configure the handler pipeline via a configuration file that is read when the framework is started. Different handler pipelines can be configured for each deployed Web service.

2.2 Workflow Execution

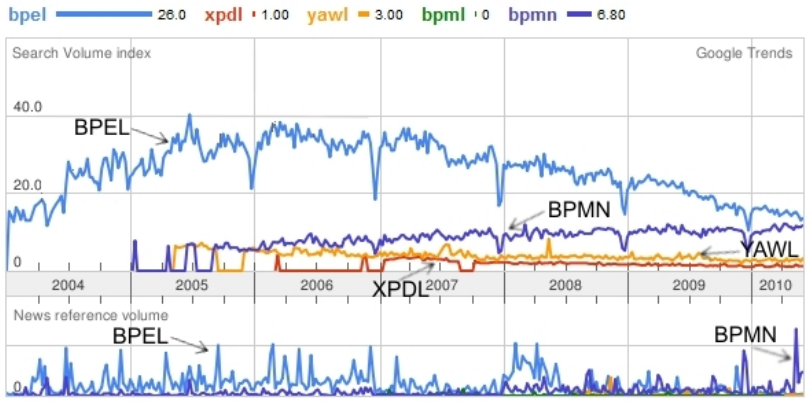
In modern enterprises, various tasks are supported by information technology. Information systems store data about e.g. vendors, customers, or products that is required in different business processes. Thus, enterprise information systems comprise a variety of different applications, tools, and databases. Small enterprises mandate IT specialists to run such systems and bigger enterprises even run an IT department. During the past years, the external hosting of services at specialized service vendors has become increasingly popular.

Today, the integration of such external systems is a very important task to offer a homogeneous working environment to employees. Therefore, the IT landscape is often designed as service-oriented architecture. The integration of services in multi-purpose business processes that require the combination of multiple services are realized as workflows. Today, many different workflow modeling standards exist like the *Business Process Modeling Language* (BPML) [122], the *Business Process Execution Language* (WS-BPEL) [75], the *Business Process Modeling Notation* (BPMN) [105], the *Yet Another Workflow Language* (YAWL) [125], or the *XML Process Definition Language* (XPDL) [50]. However, not all of these standards are capable of describing workflows in a machine executable way; For example, BPMN specifies only a graphical notation but no interpretable file format that could be used for execution. Such languages are rather used for modeling business processes on a functional instead of an technical level.

Figure 2.3(a) shows the importance of the above presented modeling languages derived from the search volume index at the Google search engine. The figure underlines the assumption that BPEL is the de-facto standard for workflow execution. The other workflow languages are only of minor importance. Additionally, it shows that the the BPMN has gained more importance in the past years as it is very popular to model workflows on a functional level while BPEL is used for the technical realization. We also apply these two standards for the architecture realization and the evaluation.

The importance of SOAP based Web services for modern IT infrastructures is depicted in Figure 2.3(b). Since the end of 2007, the REST protocol has evolved to become as a second standard for Web service communication.

We focus on the orchestration of SOAP based Web services, because today's services are mostly used for information system integration, especially in the scientific and the business domain. Modern Grid middlewares offer their services as SOAP based Web services and all commercial big players sell workflow engines that are based on SOAP services.



(a) Relevance of different modeling languages for workflows

(b) Relevance of SOAP and REST Web Service

FIGURE 2.3: Google Trends from 14.06.2010

2.2.1 WS-BPEL

WS-BPEL is an XML-based workflow description language designed for the orchestration of SOAP-based Web services. This standard focuses on the pure orchestration and ignores issues like security or versioning. A good overview is given in the WS-BPEL Primer [102] and in [79]. In the following, we present some background information and describe the most important WS-BPEL elements.

WS-BPEL 2.0 [75] is the second version of the original BPEL4WS [7] standard that was firstly published in July 2002 as a joint work of IBM, Microsoft, and BEA. It is inspired by the companies' internal workflow languages WSFL (IBM) and XLANG (Microsoft).

One of the goals of BPEL4WS [90] was to create a new XML based workflow description language that uses Web service interfaces (WSDL) only, meaning that all involved services are described with WSDL but also the workflow itself offers a WSDL interface. In May 2003, version 1.1 was released and nearly at the same time submitted to the OASIS technical committee. WS-BPEL 2.0 became approved as an OASIS standard in March 2007.

The WS-BPEL standard focuses on the creation of an XML file that can be interpreted by a WS-BPEL compliant workflow engine. This XML code is hard to read and to understand for humans without any graphical representation. Hence, often graphical editors are used for workflow modeling; for instance, the Netbeans BPEL editor presents WS-BPEL similar to the BPMN workflow modeling language.

WS-BPEL offers various activities for manipulating data, defining the data- and control-flow, reacting on faults and events, or validating messages. In the scope of this thesis, we only describe the most important structures and features of WS-BPEL and omit less important details.

Generally, a WS-BPEL process consists of partner links (described below), variables, and the actual business process that is composed of several activities. These activities can be executed sequentially or in parallel. Thereby, we distinguish **basic** and **structured** activities.

Basic activities are simple activities that interact with the outside world, manipulate data stored in process variables, generate faults, terminate the process, or just wait for some time. The most important activities for modeling workflows are the interaction activities `<receive>`, `<reply>`, and `<invoke>`. When executing a `<receive>`-activity the process waits for an incoming message that instantiates a new process or provides information to a running process. A `<reply>`-activity is used to send a result to a client. A combined `<receive>` and `<reply>` – using the same partner link and operation – models a synchronous communication between a client and the business process. The answer of an asynchronous communication can be modeled as an `<invoke>`-activity using a one-way-operation. `<invoke>` is also used to invoke an asynchronous or synchronous external service depending on the message exchange pattern. The `<assign>`-activity is used for manipulating variables, preparing messages, processing XSLT transformations, or applying relational, arithmetical, or logical operations. These activities are mandatory activities to model a basic workflow.

Structured activities determine the control-flow of a workflow. In WS-BPEL, activities that should be executed sequentially are placed inside a `<sequence>`-tag (see Figure 2.4(a)); activities that should be executed in parallel inside a `<flow>`-tag (see Figure 2.4(b)). Inside a flow, an execution order can be defined through `<link>`s. Such

links connects activities and is traversed when the source activity is finished and an optional transition condition evaluates to true. It is also possible to nest structured activities arbitrarily.

The WS-BPEL `<if>`-activity is the equivalent to an if-statement in most programming languages including the possibility to formulate a single conditional branch (if) or multiple branches (if-else and if-elseif-else). For modeling loops, BPEL provides a `<while>` and a `<repeatUntil>` operation.

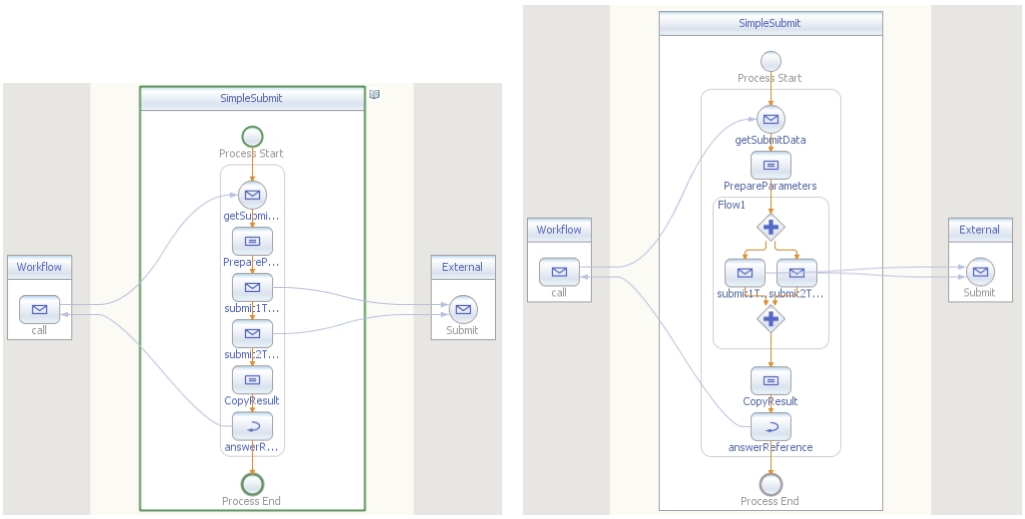
WS-BPEL provides the `<scope>`-tags to structure several activities to a semantical section of the overall workflow. As well as the whole process can have extra handlers for compensating faults or reacting to some optional events, such elements can be also attached to scope elements. Furthermore, it is possible to define variables limited to a scope.

BPEL also provides some mechanisms to deal with faults during workflow execution. They can be cached by special activities that allows for reacting with further BPEL code. Furthermore, the compensation of previously executed service calls can be modeled with compensation handlers attached to activities or scopes. The possibility for compensation enables the support for long running transactions, because already executed ACID transactions can be rolled back and e.g. a retry of the complete process scope is possible. All these fault mechanisms apply only to business faults. Exceptions thrown in the technology stack are not covered by WS-BPEL and result in a crashed workflow.

Partner links are a central concept in WS-BPEL. All communication partners, clients, services, and the workflow itself, in a WS-BPEL process are modeled as such links. Therefore we distinguish two kinds of partner links, for **invoked partner** and for **client partner** [79]. Each process needs at least one client partner link that is used to create and start the WS-BPEL process. Normally, it also has at least one invoked partner link.

As already described above, one goal of BPEL is that all workflow operations are offered by Web service interfaces using the WSDL for interface description and SOAP as communication protocol. This allows the hierarchical composition of workflows since workflows can be invoked from higher-level workflows. This enables reusability of low level business processes and a simpler modeling of complex high level workflows.

Figure 2.4 shows two simple workflow examples that are modeled with the Netbeans BPEL editor. Both invoke the same service twice, sequentially (Figure 2.4(a)) and in parallel (Figure 2.4(b)). The partners of a workflow are placed on the right (invoke partner) and left (client partner) border. The middle lane shows the actual workflow logic (stored as WS-BPEL code) as BPMN that provides standardized symbols for different activities. Both workflows are synchronous because they start with a receive and end with a reply activity.



(a) Sequential Workflow (b) Parallel Workflow

FIGURE 2.4: A simple workflow example

A business processes has a state by nature, i.e. the current execution progress, process variables, and partner links. WS-BPEL provides workflows as stateless Web services. Standards for stateful Web services like WSRF [103, 14] are out of the scope of the specification. Instead, WS-BPEL uses a self-designed mechanism to map messages to the corresponding workflow instances, so-called **correlation sets**. Correlation sets compare some messages data with some variable to find the corresponding instance. This requires that some unique data is stored in each workflow and that the invoker puts this information into the message to address the workflow instance.

The WS-BPEL standard includes an extensibility mechanism to introduce new elements - with new semantics - within a WS-BPEL process [75]. One example for an WS-BPEL extension is BPEL4People [101] that introduces new activities for integrating human tasks in WS-BPEL workflows. The extension mechanism is handy if new functions are required but it has one major drawback: The WS-BPEL engines have to be extended to support such new activities, otherwise the process is not executable. This creates a direct dependency between the WS-BPEL processes that use this extension and the extended WS-BPEL engine and thus the workflow is incompatible to other WS-BPEL compliant workflow engines.

2.2.2 WS-BPEL compliant workflow engines

To find the most appropriate WS-BPEL compliant workflow engine, we here compared several implementations. An overview concerning some basic WS-BPEL engine properties, such as License, supported BPEL version, development project status, availability, and target system is given in [62]. The report served as basis to find a suitable WS-BPEL workflow engine as basis for the orchestration architecture. This section again outlines an updated version of the report. The final selection of the WS-BPEL workflow engine is discussed in Section 4.1.1.

The overview [62] includes a list of fifty-five Workflow engines used for Web and Grid service orchestration. Several of these engines do not support WS-BPEL and are therefore omitted in the deeper analysis. At the end, [62] compares nine WS-BPEL compliant engines with respect to the feasibility to build a flexible Grid and Cloud workflow engine.

In the subsequent sections, we sum up the main characteristics of the *ActiveBPEL* workflow engine – that was finally selected –, the *Apache ODE* workflow engine – that was also a acceptable candidate –, and the *Sun BPEL Engine* that is also feasible but was not available when we did the first evaluation. The following criteria are used for comparision:

1. License - What kind of license is used for publishing the BPEL engines source code? It need to be open source.

2. Platform - What prerequisites are necessary to deploy the BPEL engine?

3. Features - What additional features are provided beyond the actual workflow execution?

4. Tooling - What tools for debugging and administration are available?

5. Community - Is the community still active?

Generally, all WS-BPEL engines that are standard compliant should be able to execute the BPEL code created in an arbitrary editor. As the WS-BPEL standard excludes suggestions for a consistent deployment, each engine has its own deployment mechanisms. Normally a deployment package is used that contains all necessary information for the deployment and the provisioning of the workflow as Web service. Such a deployment package includes all resources like WSDL oder XML Schema files. Furthermore, some engines, like ActiveBPEL, use a central file – the so-called deployment descriptor – to define additional configuration parameters such as endpoint references for partner services, persistence settings, or versioning information.

ActiveBPEL

The ActiveBPEL workflow engine used to be an open source project initiated and hosted by ActiveEndpoints. The company provided the freely downloadable source code on its homepage and gave basic user support. At the same time ActiveEndpoints sold a commercial version of the workflow engine with additional features. In 2007, the engine was under massive development and the first choice regarding its features, available support, documentation, and the already existing acceptance in the Grid domain (see e.g. [39]).

From 2007 to the end of 2008, the engine migrated from version 3.1 to version 5.0.2. ActiveBPEL integrated several new features such as the support of human interactions according to the BPEL4People standard [101]. But the company changed its business strategy and begun the distribution of a new product called **ActiveVOS**. In general, ActiveVOS is the commercial version of the ActiveBPEL workflow engine. But from then on, all new features were published under a closed source license.

For a short period, ActiveEndpoints still provided the open source version of ActiveBPEL on its homepage but in 2009, they removed the version 5.0.2. From then, only the older ActiveBPEL version 2.1 has been available on the ActiveVOS homepage. That engine is not WS-BPEL 2.0 compliant and it only supports the older version BPEL4WS.

Today, the latest open source version is again available on SourceForge[1] but has no active community. The sourceforge project website does not provide any documentation nor user tutorials but there are still some copies of the original documentation and guides available on the web. These sources are the basis for a successor project, called "bpel-g"[2] that develops a WS-BPEL engine for the Apache ServiceMix Enterprise Service Bus.

Table 2.1 shows the properties of the ActiveBPEL 5.0.2 version, as it is published on the sourceforge website, according to our evaluation criteria. The commercial version offers additional features that are not listed.

The engine demands a deployment package that contains all necessary information in a proprietary format. One part of the deployment package is the *Deployment Descriptor* that includes the actual service endpoints and workflow specific configuration parameters for the workflow engine. The basic version only supports a workflow management based on the file system as workflows are moved to or deleted from a dedicated deployment folder. After the successful deployment, the workflow is offered as service whose name is also defined in the descriptor file.

Beside this, the engine is shipped with a *BPEL Admin servlet* that provide additional functionality. It offers a simple-to-use web site that allows for engine configurations such

[1] http://sourceforge.net/projects/activebpel502/
[2] http://code.google.com/p/bpel-g/

License:	GNU General Public License (GPL).
Platform:	The engine can be deployed in an Apache Tomcat 5.5.28 service container.
Features:	ActiveBPEL 5.0.2 is completely WS-BPEL 2.0 compliant. It optionally offers workflow persistence if a database is available. The deployment descriptor offers possibilities for a service binding at deploy-time and some configurations concerning the used handler pipeline for external service invocations. The latest version of the engine offers services that implement the WS-Human Task standard [1], but without documentation. This makes the feature unusable for untrained users.
Tooling:	ActiveEndpoints provides the *ActiveVOS Designer*. This is a graphical editor to model WS-BPEL workflows and to create the matching deployment descriptor. Since version 5 the editor also supports means for implicitly modeling human tasks and the integration into the workflows. But today, the license a only temporally limited evaluation license. Hence, the ActiveBPEL engine does not provide a free editor, today. Workflow modeling is still possible with other editors but the deployment descriptor must be created by hand. The ActiveBPEL workflow engine offers Web interfaces for deployment and monitoring of workflows. Furthermore, a web page presents the most important configuration and monitoring information. It shows all deployed processes, active workflows, and a detailed process log including fault information as well as web service interfaces to retrieve this information via SOAP calls.
Community	dormant

TABLE 2.1: ActiveBPEL 5.0.2 Overview

as logging and persistence. Additionally, it supports the debugging of workflows since it produces includes a deployment and debugging view. Another service can be used for workflow deployment via a SOAP call.

Sun BPEL Engine

The Sun BPEL Engine is a service engine (SE) for the OpenESB[3] implementation. The basic components of an Enterprise Service Bus (ESB) are a message router, service engines, and binding components [29]. The router is responsible for message transformation

[3] https://open-esb.dev.java.net/

and delivery to services inside the service bus.

Service engines (SE) are software components that are deployed directly in the service bus and provide some kind of business logic. Examples for service engines are BPEL SE for workflow execution and JavaEE SE for hosting of JavaEE programs. Binding components (BC) are used to connect external services to the ESB (e.g. HTTP BC, REST BC, Email BC).

The OpenESB is compliant to the Java Business Integration (JBI) specification [128] - JSR 208 is the basic specification for the realization of a Java-based enterprise service bus. Hence, all components can generally be deployed on every JBI compliant ESB framework. The OpenESB runs in a GlassFish application server. It is perfectly integrated into the Netbeans Editor (if the SOA plug-in is installed). The open source community, mainly driven by Sun, offers a lot of general documentation as well as some video tutorials for designing workflows with Netbeans.

For the deployment of a BPEL process, a so-called service assembly that is part of the JBI specification, is necessary. Such an assembly includes information about the BPEL code, the services that are consumed during process execution, and the configurations for the required BC. It is somehow comparable to the deployment package of ActiveBPEL as it includes general information to deploy a workflow, but it has no central configuration file. The workflow service URL can be chosen arbitrarily with respect to the port and the URL path.

The engine is capable of invoking external services that are not directly deployed in the OpenESB by using the BCs. The configuration of the BCs is part of the service assembly.

Table 2.2 lists the properties of the Sun BPEL Engine regarding to our criteria.

Apache ODE

Apache ODE[4] (Orchestration Director Engine) is the WS-BPEL engine developed in the Apache foundation. The former basis for the project was the Process Execution Engine (PXE) from FiveSight which has been bought by Intalio. The engine is well documented on its website.

The user has to create a process artifact for workflow deployment. This artifact contains the Apache ODE specific deployment descriptor beside the WS-BPEL, WSDL and schema files. There are no Web service interfaces provided. For the deployment of an

[4] http://ode.apache.org/

License:	Common Development and Distribution License (CDDL)
Platform:	The engine is part of the OpenESB that is shipped within the GlassFish container.
Features:	The Sun BPEL Engine is WS-BPEL 2.0 compliant. It is fully integrated into the OpenESB though guaranteeing interoperability with various ESB internal and external services. All service endpoints are taken directly from the WSDL files of the invoked services that are included in the assembly package.
Tooling:	The Sun-BPEL-Engine SE is partly configurable within the GlassFish administrator web interface. The Netbeans Rich client provides a WS-BPEL editor that can also be used for deployment of service assemblies and interactive debugging session. A deployment, debugging, and monitoring without Netbeans is possible but not documented.
Community	active

TABLE 2.2: Sun BPEL Engine Overview

artifact, it is simply copied to the deployment folder of the engine. Nevertheless, the engine provide some Web services that enables management tasks such as the activation and deactivation of workflows, retrieving a list of all processes, process instances. On instance level, the user can ask for process variables and the event log. Thus, workflow monitoring is possible.

Table 2.3 gives the overview of this engine.

2.3 Grid Computing

The term "Grid" originates from the idea to make access to compute power as easy as to the electric power grid. It is mainly used in eScience. An early definition of Grid is presented by Ian Foster and Karl Kesselman in [80] and refined in [81]. According to Forster and Kesselman, the idea is to provide a middleware that enables inexpensive, dependable, consistent, and pervasive access to high-end computational capabilities. This should enable a new cooperation model that is called virtual organization (VO). Members of a VO do not necessarily need to belong to the same affiliation but can still work together in projects even across enterprise boundaries. A VO defines clearly and carefully what resources (compute or data resources) are shared and who is allowed to access this resources (cp. [49]).

Nowadays, a Grid environment has a typical architecture. A high-end compute resource is offered to the VO members through a so-called Grid middleware. Such a middleware

License:	Apache License 2.0
Platform:	The engine can be deployed into either a Tomcat web server with installed Axis2 framework or into a JBI compliant service bus.
Features:	The Apache ODE engine is not fully WS-BPEL compliant. There are some open issues e.g. with the `<receive>`-tag as only allowing message variables, or multiple start activities. All open issues are listed on the website. Nevertheless, it provides some other useful features such as process versioning and persistent execution of workflows. All process information is persisted in the databased and can be retrieved through a Web service. Furthermore, Apache ODE implements several WS-BPEL extensions, like "Activity Failure and Recovery", "Iterable ForEach", etc.
Tooling:	Apache ODE provides no tooling for workflow modeling; for workflow deployment and monitoring a Java client API is available.
Community	active

TABLE 2.3: Apache ODE Overview

offers services to allocate compute resources or access data resources. Modern middlewares use the advantages of a service-oriented architecture and offer these functions via SOAP-based Web services. To guarantee security and the traceability of Grid activities each user needs a personal certificate (normally a X.509 certificate issued by a commonly trusted institution) for communicating with Grid services. Thus, it is possible to map each activity directly to a natural person and formulate access rules that are based on identities. Another important Grid feature, especially for workflow execution, is the possibility to delegated rights to another Grid component, e.g. the grid middleware. This enables the middleware to execute operations on resources on behalf of the user. The user can define the delegation rights in a fine-grained way.

A Grid is composed of several Grid sites that offer different kinds of resources to the users as shown in Figure 2.5. In this cooperation, each site is maintained by its own independent administrative domain. The distributed Grid infrastructure is established by using a common VO management system. Such a system allows the administration of members, their VO memberships, and user group attributes inside VOs.

The Grid sites decide what VO or more fine-grained who is allowed to access their Grid services. Grids often provide a central monitoring and registry services to discover resources or to retrieve more detailed resource information, like installed libraries, or length of the job queue. This enables the user to find the most appropriate resource for a particular task. The detailed architecture of a Grid depends on the used middleware, the additional deployed systems and tools, the participating Grid sites, and the rules for re-

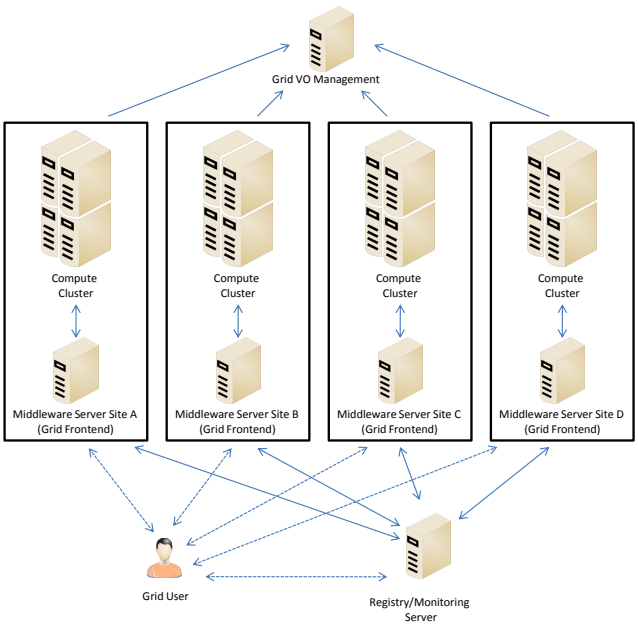


FIGURE 2.5: Example Grid infrastructure

trieving a valid Grid certificate.

Typical Grid middleware services offer the possibility to submit jobs (e.g. described in an XML-based job description language such as JSDL [8]), process file transfers (from the client to a Grid storage, vice versa, or between two Grid sites), but generally it is also possible to host self-implemented services. For submitting jobs to a cluster, the middleware communicates with a scheduler that manages the actual job execution on the worker nodes in the cluster backend. In the following subsections, the WSRF specification for Grid services and two famous middlewares

2.3.1 Web Services Resource Framework

Web services are normally stateless services. This means that the service stores no information about calls and acts the same way, each time it receives the same input data. Each call is independent from the previous and successive one. But in some scenarios, Web service calls are not really independent as the successful execution of a task needs the invocation of several different operations on one (stateful) service. This problem is tackled with the introduction of stateful Web services. A simple solution is the implicit management of instances, so that each massage includes a service instance ID to correlate the messages within the Web service implementation. Often, the correlation information

is part of the message payload. The WS-Addressing specification [23] addresses this lack and defines Web service instance addressing in a standard way.

Since Grid Jobs are often long running jobs with several state information such as the progress, the allocated resources, the job description, the start, end, wall time, and other state information, the Grid community facilitates a standard to model such stateful Web services: the Web Services Resource Framework (WSRF) [103, 14]. Since WSRF is the standard for realizing modern Grid middlewares, we call such services **Grid services** in this thesis. Beside the WSRF-compliance, Grid services require some kind of advanced security support.

WSRF is a bundle of five different standards that specifies so-called *resources* as instances of a Web service. State information is given as *resource properties*:

1. WS-Resource [52]: A WS-Resource as an instance of a Web service. A resource is addressable by an endpoint reference according to the WS-Addressing [23] specification. As an example in UNICORE 6, a resource is addressed via an unique ID that is attached as parameter `?res=<id>` to the service address URL.

2. WS-ResourceProperties (WSRF-RP) [53]: The definition of resource properties as the state information that distinguishes two instances of the same service. This includes standard operations to retrieve, update, and delete state information.

3. WS-ResourceLifetime (WSRF-RL) [115]: This document standardizes the concepts and operations for controlling the lifetime of a resource. Normally, a resource has a limited lifetime after which it is destroyed and not reachable any longer. The user can extend the lifetime by setting a new termination time.

4. WS-ServiceGroup (WSRF-SG) [95]: WS-ServiceGroup enables the grouping or aggregation of Web services and resources that are defined according to the WSRF-RP standard. This is necessary to manage WS-Resources in a registry and to enable the querying of complex requests. The belonging to a service group is defined through rules.

5. WS-BaseFaults (WSRF-BF) [91]: This specification defines the base fault type that is used to inform users about basic faults during the interaction with WSRF-based Web services.

The WSRF standards are implemented in various Grid middlewares like Globus Toolkit 4 and UNICORE 6 (see following sections) but also in the Apache Muse 2.2.0 Web service framework. The different frameworks are all based on different technology stacks with advantages and disadvantages. A performance comparison is given in [84].

2.3.2 Globus Toolkit 4

The Globus Toolkit 4[5] (GT4) is a worldwide known Grid middleware that is used in a multitude of different Grid projects. The system is generally designed as a Web service framework, offering services (e.g. WS Grid Resource Allocation and Managament (GRAM) services for compute job submission using the Resource Specifiaction Language (RSL)), a resource discovery and monitoring service (MDS), and client tools that can communicate with these services. GT4 uses the SOAP communication protocol and is based on a modified Apache Axis 1.2[6]. This is extended with implementations for the WSRF standards. Some GT4 services are also available via other protocols, e.g. GridFTP for up and downloading files to/from a grid site. It is possible to implement new services and deploy them into the GT4 Grid middleware.

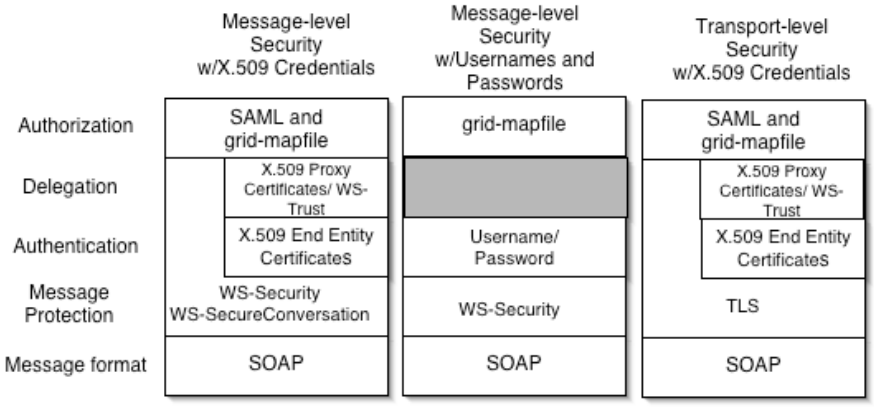


FIGURE 2.6: GSI Security Overview [129]

The GT4 security infrastructure is named Grid Security Infrastructure (GSI) and uses X.509 certificates (or very rarely user name/password) for authentication. A certificate is data that uniquely validates the owner and additional properties of a public key. The owner is identified with a unique string, called the user's distinguish name (DN).

The GSI supports three different security stacks [129] for authorization, credential delegation, authentication, and message protection. Figure 2.6 shows the three stacks including message-level security (MLS) – MLS encrypts part of the SOAP message – or transport-layer security (TLS) – TLS uses an encrypted transport channel. MLS offers more flexibility if messages must be routed via several Web service hops but encryption and signature on the basis of SOAP message content costs performance [110].

[5] http://www.globus.org/toolkit
[6] `http://ws.apache.org/axis/`

One basic concept in GSI is the usage of proxy certificates, a self-signed certificated issued by the user and signed with the users's permanent Grid certificate. Such certificates can be used for credential delegation – delegating some of the user's privileges to another entity. The set of privileges is defined by so-called restrictions in the proxy certificate.

The left and middle stack in Figure 2.6 show communication using MLS. Hence encryption and signature are done on basis of the XML message and not on the transport layer. Standards for MLS are WS-Security and WS-SecureConversation [85, 86]. The user name and password authentication, without certificates is rather unusual and prevents the possibility of credential delegation. For example, the D-Grid, German Grid initiative, does not allow user name/password authentication.

The right hand stack shows GSI using transport layer security. The user utilizes his permanent Grid certificate to connect and authenticate against the server. Delegation is possible via proxy certificates.

Security configuration (choice of the stack) can be done separately for each service. The clients need to know which security stack is required.

Server-side access control information is stored in the the so-called *grid-mapfile*, a kind of table that entries map certificate's distinguished names to local user accounts. If a user successfully authenticates with the server and an entry for this user exists in the grid-mapfile, access to the resource is granted.

2.3.3 UNICORE 6

UNICORE is a Grid middleware mainly developed at the "Forschungszentrum Jülich". It is based on Web services technologies since version 6, released in 2008. Technically, it builds up on a Jetty servlet container[7] and the last version of the Web service framework XFire[8] that has evolved into Apache CXF[9]. On top of this software stack, a lightweight WSRF layer (called WSRFlite) is created that enables the management of stateful Web services compliant to the WSRF [14] standards.

UNICORE 6 is composed of three components: the *gateway*, the *extended UNICORE user database* (xUUDB), and the *UNICORE/X* service container. Each of these components is a standalone program that communicating via Web service means with the other components. Thus, it is possible to run each component on a different machine. Figure 2.7 shows a typical UNICORE 6 environment. Normally, it provides a single point of entry, the Gateway that is installed only once. The Gateway knows all participating Grid sites

[7] `http://jetty.codehaus.org`
[8] `http://xfire.codehaus.org`
[9] `http://cxf.apache.org/`

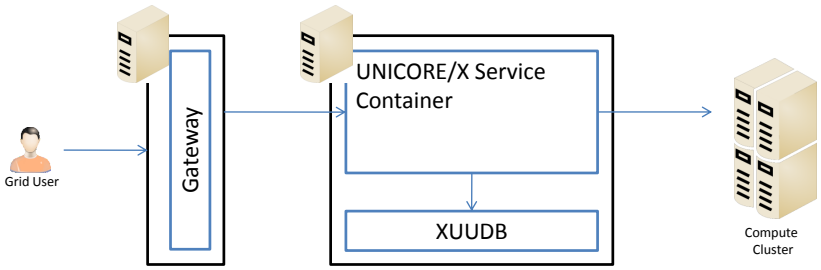


FIGURE 2.7: UNICORE architecture overview

and dispatches messages to the sites that names are part of the message destination URL. On each site, a UNICORE/X service container is installed that serves as a hosting environment for the *UNICORE Atomic services*. These services provide basic functionality for typical Grid tasks such as compute job submission and file transfers. The UNICORE 6 middleware supports several job description languages, like the Job Submission Description Language (JSDL) [8].

The user database (xUUDB) manages a table of user entries, storing the users' distinguished names, the local user names, and some basic role information. If the xUUDB is used by several sites, there must be one entry for each site the user has access to. The role information is rather some kind of user/admin role distinction than a role-based authorization system, as the roles for each user are fixed and not session dependent. As a Grid is a federation of several independent Grid sites, all of these have different access policies. Hence, the sites normally administrate the xUUDB entries locally instead of sharing a single instance that keeps control over all resources.

To submit a job to the back-end cluster, the UNICORE/X uses a component called Target System Interface (TSI). It is an interface between UNICORE 6 and the cluster scheduler that manages the job execution on the backend machines. UNICORE 6 is implemented in Java. TSIs can be implemented in an arbitrary programming language.

Security

As already stated, the Gateway is the single point of entry to a UNICORE 6 based Grid. To communicate with the Gateway, the user needs a trusted certificate (issued by a trusted certificate authority (CA)). UNICORE 6 uses TLS to protect message information. The Gateway blocks all communication except if the user presents a valid certificate. The user checks the identity of the Gateway and vice versa. The communication between all other UNICORE 6 components is secured the same way to guarantee a trustworthy relationship between these components. When forwarding the message to the Grid site,

the Gateway attests the identity of the user with a signed SAML [33] assertion. Such an assertion states that the Gateway guarantees that the user has authenticated himself correctly. The Security Assertion Markup Language (SAML) is an OASIS standard for XML based authentication and authorization data exchange. Hence, the site can be sure about a user's identity and gets the user's distinguished name from the assertion. This name is used as an id for querying the xUUDB.

UNICORE 6 adopts the OASIS standard eXtensible Access Control Markup Language (XACML) in version 1.1 [41] for authorization. This standard describes a mechanism to make an authorization decision with information from different information sources.

An XACML request contains the sections:

- The **subject** contains information about the user.
- The **target** defines the target resource that should perform an action.
- The **action** reflects the operation that should be performed on the target resource.

The UNICORE 6 default configuration provides the following information in such a request:

- Subject: The user's distinguish name, the user's role retrieved from the xUUDB, and the consignor of the message. The consignor only different from the users distinguish name if the call uses credential delegation.
- Target: The service instance that is addressed in the message. A stateless service is described by the service name. For stateful services, also the resource identifier and instance owner are added.
- Action: The operation name that is targeted with the actual message.

The request is internally sent to the Policy Decision Point (PDP) where it is evaluated against the deposited XACML policies. An XACML policy consists of one or several rules that are evaluated if applicable. Each rule contains of two sections that define the applicability of the rule:

- Target: It contains one section for matching to the XACML request: Subjects, Resources, Actions. In each section, matching rules are defined, like "the distinguish name of the subject have to be equal to a fixed distinguish name", or "the resource-id of the target resource must be equal to a fixed service name". Therefore, XACML provides a large set of comparator operators for basic but also complex data types that are typically used in an authorization process. These rules are limited to the information of the respective section.

- Condition: This section is used to define conditions between the user, the resource, and the action. As example, it is possible to formulate the following condition: ”evaluate to true if the subject’s distinguish name equals the resource owners distinguish name”.

Only if both parts and all the included functions evaluate to true, the rule is applicable. Each rule evaluates either to a permit or a deny result. XACML offers means to combine several rules to a policy.

As already described, UNICORE 6 uses SAML assertions to confirm the authenticity of a user. For credential delegation, it also utilizes the SAML standard. The client that wants to delegate rights to another entity (another client or service) attaches a signed SAML assertion to the message he sends to the second entity. This delegation assertion includes – among others – the DN of the user and the DN of the second entity. The latter can use this assertion to prove the delegated rights to athird party. It is possible to delegate all but also only a well defined portion of rights e.g. only the right to execute a single operation on an external service.

UNICORE Virtual Organization Service

The xUUDB provides only very basic user attributes such as a single role and the local user login. For managing VOs, where a single person can participate in several VOs and hold different roles in each VO, a more sophisticated user attribute management system is necessary. The UNICORE 6 team developed a VO management system called UNICORE Virtual Organizations System (UVOS) [18] – within the Chemomentum project[10].

This system can be integrated into each UNICORE 6 installation and substitutes the xUUDB as information provider for the authorization process. In UVOS, users are managed as entities to those administrators can attach arbitrary attributes. Such entities can become member of hierarchical organized groups that can be seen as VOs. The administrator of a group can grant group administrator rights to other entities so that there can be different administrators for each group. UVOS distinguishes three kinds of attributes [19], **global attributes**, **group-assigned attributes**, and **group-scoped entity attributes**. Global attributes are directly attached to an entity and are always valid. Group-assigned attributes are attached to a group and all members of this group automatically inherit the attribute. Group-scoped entity attributes are again assigned directly to an entity, but are restricted to a special group and are invalid outside the context of this group.

[10] `http://uvos.chemomentum.org/`

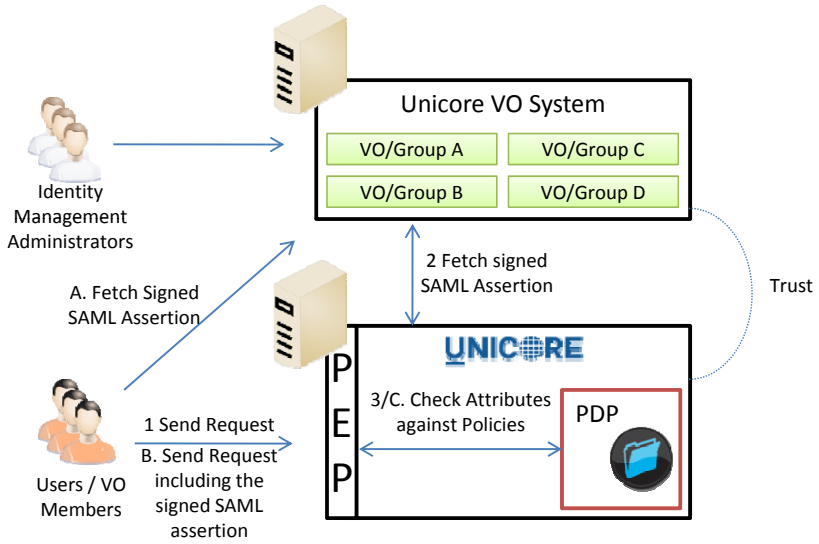


FIGURE 2.8: UNICORE Virtual Organization Systems - Push and Pull Attribute retrieval

Those attributes can be retrieved as SAML assertions in two ways, **push** and **pull**. Both ways are depicted in Figure 2.8 (1-3, A-C). Generally, the responsible identity management administrators maintain the user entities of their VO – adding or removing users from the VO, partitions them in subgroups, and assign attributes. These tasks are independent from querying the information.

The two mechanisms to retrieve attribute information from UVOS demand additional effort for at least the Grid site administrator. When using the push mechanism (A-C), the user is responsible for fetching the SAML attribute assertion from the UVOS server. This way, he can control what attributes should be included in the assertion by defining the context situation for the call. This offers data security and data minimality because each site cannot request all information about the user.

In the pull scenario (1-3), the UNICORE 6 middleware requests the assertion for the current user from the UVOS server. The server does not know the context of the current call and therefore requests all attributes. It is possible to restrict the access to user attributes to some VOs but inside these limitations the server can ask for all attributes, even if they are not important for the current call.

Both ways result in a number of user attributes available during the authorization process that is triggered at the UNICORE 6 Policy Decision Point (PDP). These attributes enable the service provider to define more finer-grained access control rules for the deployed

services. This enables the possibility to introduce a role based access control mechanism that considers the user's roles in the different VOs.

In the remaining of the thesis, we distinguish the UNICORE 6 architecture not such detailed. If we talk about UNICORE 6, we regard the system as an hosting environment for stateful services.

2.4 Cloud Computing

This section introduces the idea of **Cloud Computing**, as it has been formed over the past two years. The term "Cloud Computing" is mainly driven by the industry and has been largely adopted by the research domain. The cutting-edge commercial cloud computing provider is Amazon with the Amazon Web services[11] that offer various possibilities to use compute power from Amazon's data centers. These services are all based either on storage or on virtual machines technology enriched with additional services for e.g. load balancing, monitoring, or relational databases.

The main characteristics of cloud computing products are [126, 26, 10]:

- on-demand: the user can allocates resources in a very short time and configure these resource according to his needs (e.g. main memory size or software stack).
- pay-per-use billing: the user only has to pay for resources he really allocates, often on a resources-per-time basis.
- easy-to-use: the usage of the resources is rather simple and can mostly be done without a lot of experience and knowledge about the services background.
- scalable: the user can allocate as many resources as he needs, nearly without limitations.
- minimal management effort: the system must provide an easy registration, monitoring, and billing capabilities.
- elasticity: the possibility to dynamically scale up or down in a short period of time

To maximize the utilization of data center resources, each Cloud computing product is designed as a multi-tenant architecture. This means that several customers commonly use the same hardware but without knowing anything about each other. The configuration, data, or deployment of one customer must not affect the configuration, data, or deployment of another customer. He should not even know that someone else is using the same

[11] `http://aws.amazon.com`

hardware. For this, often virtualization technologies are used to isolate the customers from each other.

Since the introduction of the first Amazon Web services, many other companies took up the idea of Cloud computing and offer Internet services that fit into the above listed criteria. Today, a diversity of Cloud services is available from hosting virtual machines to offering a complete browser-based online solution for a customer relationship management.

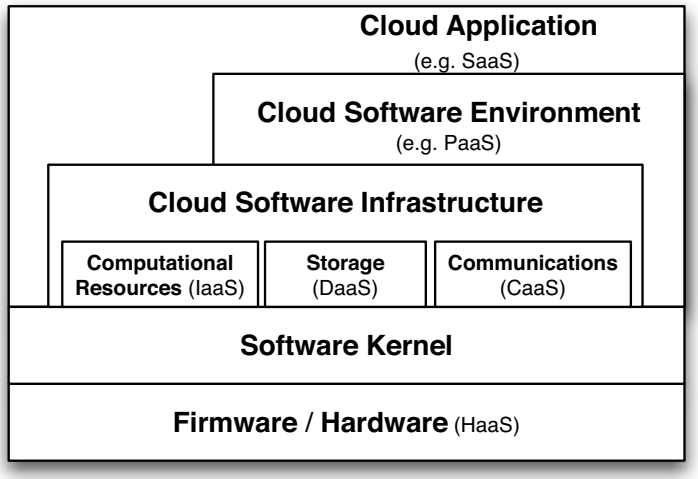


FIGURE 2.9: Ontology of Cloud computing [132]

Yousseff et al. have given a first definition of cloud computing shapes in [132]. They distinguish five layers, cf. Figure 2.9. The bottom most layer is the actual hardware in the data center, as compute resources and network, called *Hardware as a Service* (HaaS). Above this layer, the *Software Kernel*, all the software that is necessary to run a Cloud service is placed. As examples, this can be software for virtualization of the hardware resources so that each user can get his environment or a Grid middleware. On top of these two layers, the actual Cloud services that the provider sells to his customers are placed. Yousseff et al. call these layers *Cloud Software Infrastructure*, *Cloud Software Environment*, and *Cloud Application*.

This definition dealt as basis for developing a common understanding of Cloud computing definition, but the concepts has been defined slightly differently in other papers and articles, e.g. [27, 126, 10]. Today, the terms *Infrastructure as a Service* (IaaS), *Platform as a Service* (PaaS), and *Software as a Service* (SaaS) are widely accepted and also used in this thesis. Furthermore, the thesis presents *Orchestration as a Service* (OaaS) as a special

occurrence of PaaS (cp. Section 3.2). The following sections describe the main properties of these three Cloud computing shapes.

2.4.1 Infrastructure as a Service

The term *Infrastructure as a Service* (IaaS) describes all services that directly provide some kind of infrastructure to the user. This includes compute power, network, and storage. Normally, the compute resources are sold as virtual machines that the user configures according to his requirements. The images for these virtual machines are stored at the Cloud storage service where the user can download or upload nearly arbitrary amounts of data. Among these basic services, the vendor often offers additional services like network services, load balancers, or firewalls. It is possible to (automatically) scale the number of virtual machine instances according to the current load. The seamless and rapid scaling capabilities of resources are called Cloud elasticity.

On this layer, the user has to pay for the actually used resources mostly on a per hours basis. As index for the pricing model, vendors use the number of CPUs, the size of the main memory, or predefined instances with a fixed number of CPUs and main memory. Storage resources are billed by the size of the used storage and data transfer is billed by the amount of in- and outgoing data. Services like load balancer or monitoring services are billed separately also on a per-usage pricing model.

Some famous vendors of IaaS provider are Amazon[12], ServePath[13], and Rackspace[14].

2.4.2 Platform as a Service

The *Platform as a Service* (PaaS) layer offers an environment, where customers can deploy their self written programs or applications on the vendor's hardware. The advantage of this compared to IaaS is that the user does not need to care any longer about maintaining the operating system or installing the runtime environment for his application.

Each PaaS provider offers an API that supports the software developer by offering means for e.g. storing data, monitoring the application, or authentication and authorization capabilities. Of course, programming against such an API causes a strong dependency between the PaaS provider and the PaaS user because changing the vendor will result in at least reprogramming some parts of the application. Furthermore, platforms are often restricted to only a few programming languages so that the user must choose his provider carefully.

[12] `http://aws.amazon.com`
[13] `http://www.gogrid.com`
[14] `http://www.rackspacecloud.com`

Often, PaaS providers offer other features to the customers like scaling the application or cost control mechanisms.

The billing of PaaS services is based on the same criteria as IaaS. The applications are running on hardware resources that are equipped with the platform software. For instance, the ratio of hardware consumption is billed on the pay-per-usage model. Storage, network traffic, and additional services like load balancer are also billed separately.

Google provides the Google App Engine[15] as its PaaS software stack. It is a platform for Python and Java applications that are billed only on the really consumed CPU cycles – idle services are for free. As a second example, Microsoft offers a platform for .Net services, the Microsoft Windows Azure Cloud[16] that is integrated in the Visual Studio IDE. The user can test the implementation locally and can deploy it on the cloud within a few seconds.

2.4.3 Software as a Service

Software as a Service (SaaS) is the Cloud computing shape with the smallest degree of freedom. It originates from the application service provisioning. A SaaS provider runs software in its data center that is used via Web service interfaces or simply via a browser. The vendor is responsible for the complete software stack and hardware. Thus the end user does not need to handle any software updates or patches. The usage of a SaaS service is also billed with a pay-per-usage model, e.g. a fix price for each user account per month or number of stored datasets. When using SaaS, the customer saves the money for running and maintaining own hard- and software.

The provider must guarantee a very high quality of service, especially concerning availability. He has to improve his product regularly to preserve its competitiveness. The major drawback for a customer is that he depends on the SaaS provider because a migration to another SaaS provider will be costly. Often such systems lack on standard data in- and export mechanisms.

A cutting-edge SaaS provider is salesforce.com with an Customer Relationship Software for all sizes of companies.

2.4.4 Cloud Computing as innovation engine

The idea of Cloud computing enables a paradigm shift in running IT for private and business applications. Especially small and medium enterprises that are not able to finance

[15] `http://code.google.com/appengine/`
[16] `http://www.microsoft.com/windowsazure/`

an IT department can profit from using services on a Cloud infrastructure. Thus, it is possible to cheaply introduce some sophisticated software from a SaaS provider that supports business process tasks or to setup some services on an IaaS or PaaS infrastructure for offering products/information to customers. The initial fixed costs for buying hard- or software and the resulting costs for maintaining this hard or software often hamper the realization of risky business ideas. Additionally, possible projects that do not justify the purchasing of high performance computing systems, e.g. projects with only a temporary high resources demand, are now possibly realizable.

The Cloud computing products help to face these barriers with simple and relatively cheap mechanisms. Today, it is possible to offer services in a simple and scalable way without initial costs in each of the three Cloud computing layers. Each layer offers different means to provide some kind of services. This allows for using of professional software support and to offer scalable services with nearly no initial costs. The scalability of such systems guarantees quality of service so that customers are not scared by possible downtimes due to overloaded services. A famous example for this is the “Animoto” service that started with only 50 instances on the Amazon EC2 cloud and scaled up to 3,500 instances within three days after integrating the service in facebook [34].

3 Requirements for secure Workflow Orchestration

Contents

The first hype for Web services as a key technology for integrating information systems is over. Today, especially in the business domain, means like SOA and Web service orchestration have been established in a productive manner. In eScience, workflows have been gaining more and more importance during the past few years. These experiences from both domains allow us to state that SOA is a feasible concept for realizing flexible IT landscapes.

In the business domain, the integration of enterprise information systems is called Enterprise Application Integration (EAI) and is a well-established means to powerfully support business workflows on the technical system layer. EAI offers extensive benefits for the management, maintenance, and reusability of IT systems. It helps the employees to focus on their work instead of hampering with different graphical interfaces for each application, entering the same data twice, or waiting for data to be processed [108]. A fully integrated system will ease the everyday work, accelerate business processes, thus saving money and making the company more competitive.

In the scientific domain, scientists often use only small self-provided infrastructures without the support of an IT department. Compute resources are only required temporarily so that establishing local data centers in many scientific institutions is unprofitable. But the complexity of scientific problems has increased over the recent years and therewith the demand for high performance compute resources has grown. This increases the need

for new cooperations between scientific institutes to allow the operation of large high performance data centers and a collaborative usage of these resources. This paradigm for distributed computing in science is called electronic science or short eScience. eScience provides adequate means for collaborative research. It enables scientists – not only computer scientists – to access relevant resources and tools for handling complex scientific questions from the respective domains.

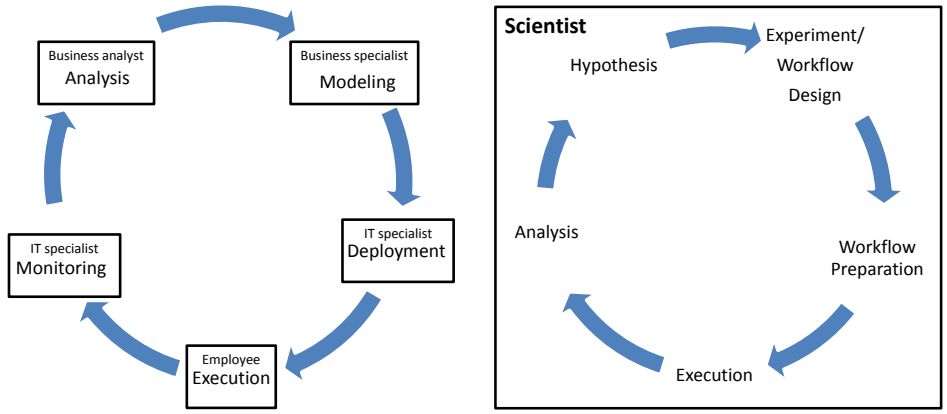
This chapter gives an overview of the two main application domains, information system integration for science and business. It describes these two scenarios and points out the differences. Furthermore, we present a new idea for information system integration called *Orchestration as a Service* that facilitates the usage of outsourced orchestration services.

3.1 Business vs. Scientific Workflows

Technologies for distributed computing like Grid and Cloud computing are used in many different application domains. Grid computing originates from the scientific community but has also been adopted by large enterprises, e.g. the Enterprise Grid. The basic Cloud computing idea is lead by the Amazon Web Services offering numerous services to customers ranging from accessing the online shop up to requesting human labor[1]. Additionally, Amazon provides compute and data storage services with thousands of customers from all over the world, mainly originating from the business domain. Recently, scientists started to explore the advantages of Cloud computing for their work and adopted Cloud technologies for eScience. In this field, Grid and Cloud technologies (especially IaaS) are partly competitive technologies, both having advantages and disadvantages regarding to different scientific scenarios and the respective requirements to the compute infrastructure.

Before defining the requirements for the Grid and Cloud services orchestration engine presented in this dissertation, we have a closer look on both scenarios especially with respect to the people and roles who are involved in workflows' life cycles. Since the orchestration engine originates from the BIS-Grid project that had an explicit focus on business workflows, the architecture will primarily target business demands, but considering scientific computing requirements as far as possible.

[1] https://www.mturk.com/mturk/welcome



(a) Business Process (b) Scientific Process

FIGURE 3.1: Process Life Cycle and Roles

3.1.1 Business Workflows

The usage of workflow management systems has a long tradition in business computing. Since the 90s, used techniques evolved and are a well established means for supporting, monitoring, and improving business processes today. Modern workflow management and execution frameworks are well tested and provide a comprehensive range of functionality. Executable workflows are used for mapping business processes to a technical system level and establishing a high level of IT support. By integrating notification systems or special Web services for human interaction, it is even possible to directly integrate employees into the workflow on this technical level.

Goal

The goal of workflow management in business computing is to enable, accelerate, and support business processes. Such business processes can be manifold, representing often an important part of the company's business core competence. Business processes are long-living. Several life cycle iterations are usual before a production-level quality of a process has been achieved. Such stable processes are often used as building block for standard enterprise tasks [15]. The productive character of business workflows is of paramount importance, as customers pay for the execution expecting a guaranteed quality of service level for functional amd non-functional properties.

Life cycle

The structure of a modern enterprise is composed of several departments. The realization as well as the execution of each business process normally requires the cooperation between a couple of those departments. Specialists continuously analyze the workflows for finding weaknesses and improving their efficiency.

The typical life cycle of a business workflow consists of five consecutive phases: modeling, deployment, execution, monitoring, and analysis. Each of these phases is conducted by some expert roles in Figure 3.1(a). Hence, each phase requires an expert who is responsible for the particular life cycle phase. Generally, we can distinguish such specialists according to their roles [113]. Analogously, each specialist uses his own highly specialized tool that requires expert knowledge of the respective domain. The following enumeration describes the five phases, the roles, and examples for the tooling:

- Modeling: A **business specialist** models the new process, e.g. using the Business Process Modeling Notation (BPMN) [105]. For typical business processes, the process control-flow is more important than the data-flow. This means that generally an activity does not start until another activity is finished [93]. If a process is modeled for the first time, it is very helpful to model the as-is state in advance as a starting basis.
- Deployment: An **IT specialist**, well trained in SOA and executable workflow modeling, transfers the process model into an executable format (e.g. WS-BPEL) and deploys it on the company‘s workflow engine.
- Execution: The **employees** or customers are executing the workflow. Usually, a company provides a client application or Web page to provide an interface for interaction with the workflow.
- Monitoring: Another **IT specialist** collects monitoring data, on the one hand for finding failures and technical bottlenecks in the workflows and the IT infrastructure. On the other hand he collects data as input for the business analysis.
- Analysis: To improve the business process, a **business analyst** examines monitored data. The focus of this analysis is on business indicators for identifying errors or non-optimal order of activities in the process.

3.1.2 Scientific Workflows

Nowadays, scientists have to deal with similar problems like companies. The growing complexity of experiments creates an increasing demand for resources and cooperation of several scientific institutes. This raises the need for a flexible integration of various

information systems, compute resources, and scientific instruments [15]. More and more scientific applications, compute power, and experimental data are becoming available as service via the Internet and scientists use these resources conjointly. They require an high quality of service especially for security, data provenance, and remotely executed computations. In eScience and in the context of Grid computing, such consortia are called virtual organizations.

As example, numerous scientists evaluate data that is collected at the "Large Hadron Collider" (LHC). The LHC experiments produce peta-bytes of data that is made available over the Internet to scientists all over the world. These scientists need compute power for processing this information in their experiments and evaluations. Beside the advantages, such as availability of the data, cooperative scientific computing raises new technical challenges such as the flexible handling of diverse data formats. Scientists often have to integrate experimental data, e.g. raw data from scientific instruments or proprietary data formats from partners. SOA and workflows provide means to handle such different data formats and services. The scientists are enabled to combine data sources and services to transform data and to collect the information from different sources and process them on many compute resources.

In the recent years, the idea to adapt workflow management systems from the business domain to model arose, execute, and monitor scientific workflows. Several papers and articles like [51, 93, 113, 15] analyzed the main characteristics of both workflow domains and stressed out the differences in requirements. [15] classified four kinds of workflows that are similar in both domains but also identified some diversity in the typical workflow life cycle.

The characteristics of stable and productive workflows can also be found in science workflows, especially if non-IT experts or domain beginners only use workflows as recurring tasks instead of modifying workflows. As example, energy minimization in molecular science experiments is a standard task that requires a fixed workflow. In contrast to such standard tasks, a lot of scientific workflows show highly dynamic properties. Scientists have to build new workflows from the scratch, modify activities from template workflows, or change the experimental design to improve results. Such activities require the scientist to be enabled to handle all workflow life cycle phases.

Goal

Ludäscher [93] describes the goals of scientific workflows twofold. On the one hand, it is to save scientists' time by enabling them to deal with domain specific aspects rather than dealing with complex data management and software issues. On the other hand,

workflows should save processing time through optimizing the utilization of available resources.

In e-Science, scientists use information systems and compute resources to confirm or invalidate their hypotheses. Since compute resources and domain specific applications become available for many scientists, many experiments are done as so-called in-silica experiments because of simplicity and cost reduction. In-silica means that the experiments are simulated on compute resources instead of real experiments. This allows the scientists to process hundreds of experiments in a short time and without costs for scientific equipment or dangerous materials. Of course, this requirements for compute and data storage resource for in-silica experiments cannot be provided by a single institute. Thus, the need for resource sharing and cooperation has become essential. This need shall be met by scientific workflow execution.

Workflows also provide the possibility to clearly describe a complex experiment and guarantee reproducibility and provenance of scientific results.

Life cycle

A typical scientific life cycle is similar to the life cycle of a business workflow, but the importance of life cycle phases differ. Figure 3.1(b) on page 41 presents a scientific workflow life cycle (cf. [93]). The already described explorative character of scientific workflows results in only few workflow iterations. For a scientist, the experimental design and analysis phases are often more important than a stable workflow execution.

The life cycle depicts the most significant difference between business and scientific workflows: A single scientist is responsible for all life cycle phases. This requires comprehensive knowledge about workflow modeling, execution, and result analysis. Hence, a scientist needs an easy-to-use but extensive tooling support.

The scientific workflow life cycle phases are the following:

- Hypothesis: The **scientist** starts with designing the experiment according to his hypothesis or experimental goals.
- Experiment/Workflow Design: To prove the hypothesis, the **scientist** designs a workflow. This includes the selection of required data, applications, and compute resources. Possibly former or third party workflows are reused, integrated, or modified.
- Workflow Preparation: This phase includes the allocation of compute resources and provisioning of data, i.e.upload data in repositories. The **scientist** must deploy

the workflow or use workflow design tools that implicitly generate an executable workflow description and make it executable on a workflow engine.

- Execution: A typical scientific workflow consumes data and produces new data. Scientific workflows represent the data-flow rather than the control-flow. Due to the explorative and long-running properties of scientific workflows, the **scientist** often requires runtime monitoring to check progress and evaluate intermediate results. Sometimes an online reconfiguration of the workflow is desirable.
- Analysis: During the data reprocessing, the **scientist** generally checks the produced data, whether it confirms the hypothesis or not. Sometimes, it is necessary to re-run the experiment with different parameters, to slightly change the workflow description, or to adjust the hypothesis and re-start the workflow life cycle.

3.2 Orchestration as a Service

A Web service orchestration engine is a complex system that is costly to setup and to maintain. Running a commercial engine can exceed the IT budget for scientific institutes as well as for small and medium enterprises. But workflow engines are a central component of modern SOAs and helpful for flexible integration solutions. As already presented, workflows are beneficial for both domains, science and business.

Scientists have already noticed the lack of resources for running an orchestration engine on their own. Therefore, domain specific VOs create powerful tools for modeling and executing workflows. Numerous examples for such projects exist such as projects for molecular chemistry, high energy physics, or astronomy. For these cooperations, the usage of Grid computing technologies is a common means.

As described in Section 2.4, Cloud computing offers technologies to use resources, infrastructures, and software on-demand that are hosted externally at a service provider's data center. SOA and Web technologies create the basis for information system integration, Grid and Cloud computing. Thus, the ideas to offer a service that provides capabilities for service orchestration in a Cloud computing fashion come into mind. Such a service shall be able to integrate various kinds of Cloud services as well as services hosted in the company's IT department. In the same time, it shall be applicable to several application domains to address a high number of possible customers. Its architecture must be built on standards to reach a high confidence level. Since different security technologies are used in Web, Grid, and Cloud services, it shall be expandable to support new Web services and security types. We named this idea **Orchestration as a Service** (OaaS).

OaaS is a central topic in this dissertation and arises additional requirements for the design of a workflow engine that should be applicable this context.

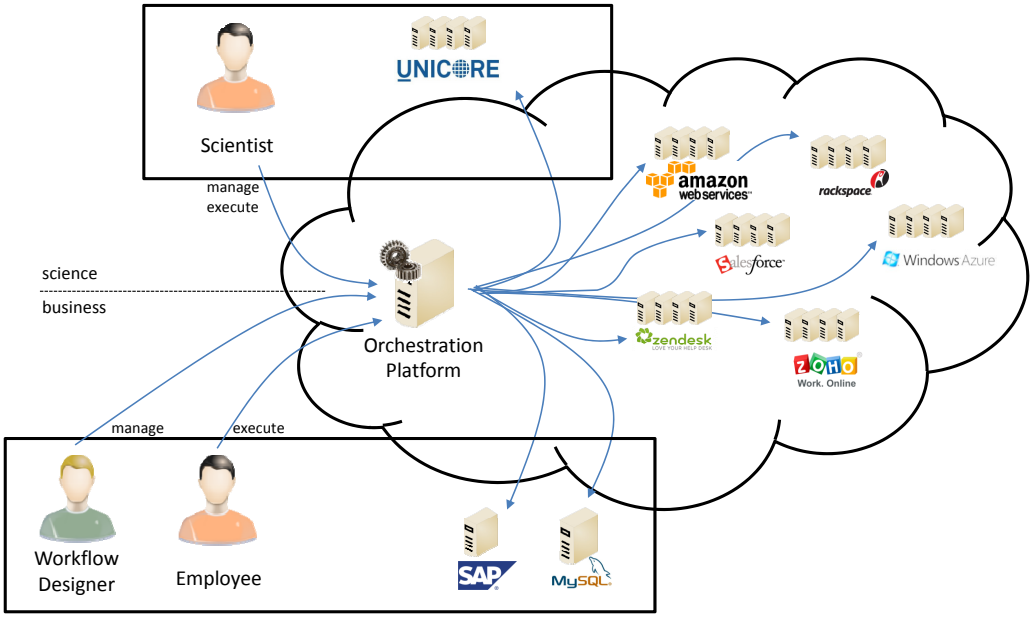


FIGURE 3.2: Orchestation as a Service scenario

Figure 3.2 depicts the general idea of Orchestration as a Service embedded in the IT landscape presented in Section 1.1 on page 3. Again, the data centers are annotated with products', companies', middlewares', and Cloud computing providers' names to illustrate the diversity of the service landscape. Generally, an OaaS provider offers means to manage self-designed workflows and execute these workflows in a secure and privacy-aware manner. Concurrently, it shall be flexible enough to support various kinds of services and security requirements. Ideally, the OaaS workflow engine is be applicable for business as well as for scientific scenarios.

With the default terms of Cloud computing, an OaaS platform can be seen as a specialized form of Platform as a Service (PaaS) from the workflow developer's view. The developer gets a platform to deploy and also to offer workflows to his users. Such workflows generate additional value for the users without revealing any internals of the workflow. They simply use some kind of Web service at a service provider. These are basic characteristics of the Software as a Service (SaaS) definition (from the workflow user's view). To make an orchestration service a Cloud service, OaaS needs to adopt the main non-functional Cloud computing characteristics as presented in Section 2.4: easy-to-use, pay-as-you-go pricing, elastic scalability, and on-demand usage [126, 26, 10].

For guaranteeing privacy and security of workflow definition, workflow execution, and workflow data, a platform for OaaS must be designed as a multi-tenant architecture. This

means that workflows must run completely isolated from each other. Furthermore, running workflow instances and even deployed workflow definitions must solely be visible for the corresponding workflow administrator and authorized customers. Since the OaaS platform shall be cheap and competitive, it is necessary to consolidate workflows of different customers on the same hardware instead of providing individual hardware to each customer. This requires appropriate mechanisms regarding access control and information filtering according to the user's identity.

Customers will benefit from OaaS. The usage of a professionally hosted Cloud orchestration service will lower initial costs for integration of in-house hosted information systems. Due to the extremely focused business concept of providing the orchestration engine, the service will reach a higher availability and reliability than self-maintained workflow engines. It is maintained by specialists, always up-to-date, and probably even a 24/7 support is available.

3.3 Requirements

To achieve the benefits listed above using orchestration services in business and eScience and to face the presented challenges, the workflow engine architecture must fulfill several requirements. The implementation of a technical solution for complex integration scenarios requires a flexible and expendable workflow engine that offers high security standards and connects to different modern Internet services.

This section lists functional and non-functional requirements that are firstly important business collaborations as well as for eScience. In doing so, we include the two following application scenarios:

- the integration of Internet-available stateless and stateful Cloud and Grid services in business processes (and eScience) where today's workflow engines only support very limited security capabilities.
- the integration of Cloud and Grid services as added value using the "Orchestration as a Service" model. Here, the workflow orchestration itself is the product that is offered to customers.

Since this work focuses on orchestration, we basically limit the requirements analysis to the technical management and execution of workflows. These tasks are mandatory for both life cycles. Topics like workflow modeling, business process analysis, and the interpretation of intermediate results are only of minor importance. Some requirements have been investigated in the scope of the BIS-Grid project and are published in the project deliverable 3.1 [62] that mainly considers the integration of Grid services for business scenarios.

The here presented requirements are based on that deliverable but we also consider new requirements arising from the OaaS scenario. Furthermore, [40, 113] provided further ideas to refine this early requirement analysis.

3.3.1 Basics

The following requirement are general and not categorized to workflow modeling or execution.

RQ1: Standards

The business domain has been developing workflow management systems for the past two decades. The in this process established methods and standards are well tested in several hundred productive environments. To increase the acceptance of the workflow management solution it shall be based on a mature fundament. In the recent years, several scientific projects have developed workflow engines as well (see Chapter 7), but these engines were tailored to different scientific application domains and hardly applicable to other domains. There is no common standard for Grid service orchestration in eScience as the de-facto standard WS-BPEL in the business domain.

Several scientists evaluated the applicability of commercial workflow management engines and standards for eScience. They try to adopt the WS-BPEL workflow language and the whole business process management systems to model, execute, and monitor scientific workflows and listed missing features for an efficient reuse of this technology (cf. [112, 61, 39, 111]).

The adoption of existing standards allows for reusing standard compliant components and the the adoption of already existing third-party tools. Additionally, such standardized components are exchangeable with other standard-compliant interfaces which partly increase system and vendor independence.

RQ2: SOAP communication protocol

SOAP is the basic technology for Web service message exchange as shown in the fundamentals (see Chapter 2). It is used in several modern Web service development frameworks like Apache Axis[2], XFire[3] and its successor Apache CXF[4]. These frameworks also form the basis for modern Grid middleware systems like UNICORE 6 (XFire, Version 1.6)

[2] `http://ws.apache.org/axis/`
[3] `http://xfire.codehaus.org/`
[4] `http://cxf.apache.org/`

and Globus Toolkit 4.0.8 (Apache Axis, Version 1.2 RC2) as well as for Cloud computing offerings and hundreds other publicly available Web services. For example the Cloud services "Amazon EC2" and the SaaS provider "Salesforce" support a SOAP interface as well as traditional Web services like the "PayPal" payment service or "eBay".

Furthermore, SOAP is a standard that matured over the past decade and is evaluated as suitable for Enterprise Application Integration and workflow processing. Several open source and commercial workflow engines are available to support SOAP-based workflow orchestration. The workflow orchestration engine developed in the context of this dissertation shall be compatible to existing EAI solutions. Hence, we demand the support of SOAP as message exchange protocol to also facilitate the communication with modern Grid middlewares and Cloud services.

RQ3: Basic Activities Support

Emmerich et al. [39] demand support for so-called basic activities to orchestrate Grid services. Here we generalize this requirement. An orchestration must be capable of invoking Web, Grid, and Cloud services and to model the control- and data-flow between such invocations (cp. RQ10). Furthermore, it must support the assembly/extraction of data to/from message content. Especially in the context of Grid computing, it must be possible to submit jobs to Grid middleware systems and to initiate third-party data transfers between Grid sites.

RQ4: Role-based Access Control

Role-based access control is a standard procedure in business IT. The access to workflows and services is restricted to company roles actually participating in the business process. The user normally authenticates him against the system with a user name and password. Sometimes, he has to choose one available role for the current session to realize separation of duties. Access rights are not granted to users but to roles.

As described in Section 2.3, Grid computing uses the concept of virtual organizations to grant access to resources. The precondition for becoming a VO member is the possession of a valid Grid certificate issued by a Grid-wide accepted Certificate Authority. Though there are also different roles in a virtual organization, the access rights to resources are mostly granted on the basis of virtual organization membership.

As a requirement for the Grid and Cloud workflow engine, we demand the ability to handle role based access control as well as certificate based authentication. Especially for usage szenarios that must enable credential delegation – as required for accessing Grid

resources from a third-party workflow engine – personal certificates for each user are a core requirement.

RQ5: Secure Communication

When executing workflows across enterprise boundaries, the cheapest solution is to use the Internet as communication platform. Solutions like VPNs are adequate to enable an end-to-end communication between partners but they are inflexible on the other side. The integration of new partners in VPNs is costly since both partners must set-up new connection endpoints. As Cloud computing offers various capabilities to deploy and use services as well as preserving the possibility to rapidly exchange a partner or to use an identical service at another provider, we need a more flexible way for communication.

Therefore, we demand directly encrypted end-to-end communication between the workflow engine, users, and invoked services.

RQ6: Integration of new service providers

Cloud services evolve quickly and the market grows steadily. To integrate Cloud services in workflows the workflow engine must be enabled to handle various security mechanisms for authorization and authentication.

This demands a flexible mechanism for integrating new security mechanisms into the workflow engine. It must be possible to deposit credentials as well as to implement support for additional security technologies.

RQ7: Hierarchical Composability

The complexity of workflows (business and scientific) can be very high. Hence, it must be possible to structure workflows in sub-workflows that can be seamlessly integrated into higher level workflows.

RQ8: Stateful Services Orchestration

Grid services are implemented according the WSRF standard that explicitly enables Web services to manage and to offer standard means for manipulating a Web service instance state. The workflow engine must be capable of orchestrating stateful services. This includes the creation, invocation, and destruction of WSRF-compliant Grid service instances as well as invoking the operations for state manipulation.

RQ9: WSRF-compliance

A state is an inherent property of workflows, for instance, the current execution progress, content of workflow variables, or involved partners. To enable the retrieval and manip-

ulation of workflow states in a standard manner (cp. RQ1), the workflow orchestration engine should provide the workflows as WSRF-compliant Grid services.

One major advantage of WSRF-compliant services is the standardized addressing of instances. The standard defines that the instance identifier must be part of the addressing information in the SOAP header. This enables the workflow engine to identify the target workflow instance before delivering the message to the workflow service. Today's workflow engines use correlation sets for mapping messages on workflow instances (on basis on message payload as identifier). These correlation sets are complex and must be defined separately for each workflow. The workflow service implementation can only determine the workflow instance after the Web service framework has passed the message to the workflow service.

The usage of an state-aware Web service framework such as UNICORE 6 allows an access decision *before* passing the message to the service. For making an access decision, the security system can also consider data from the workflow instance's state as well as the workflow owner because the instance is explicitly identified. This enables more sophisticated mechanisms for access control system and access rule definition. Additionally, the explicitly marked instance identifiers are useful for load balancing issues.

RQ10: Control-flow and data-flow modeling

Business workflow modeling focuses on the control-flow which is the coordination and structuring (sequential, parallel) of activities. In contrast, the data-flow refers to the path of input and output data between workflow activities. The control-flow is in the focus of business workflows since it describes the chain of activities in a business process. Scientific workflows are rather data-flow oriented. The availability of data at the processing nodes and the streaming of data from one node to another are more important than the separated execution of tasks.

Since, passing of large amounts of data through an orchestration engine would require extremely high capacity regarding bandwidth and compute power, the workflow engine must be capable of initiating and controlling third-party data transfers. The process designer shall be able to model both, control- and data-flow.

RQ11: Workflow Isolation/Design for multi-tenant usage

The whole system shall be designed to be used in a multi-tenant environment. This means that several customers work on the same hardware without knowing each other. For every customer it seems that he uses a dedicated instance of the service. However, modern Cloud computing services use the hardware resources in a shared way to profit from a higher resource utilization and provide the services cheaper. For this, Cloud providers employ technologies as virtualization to separate customers from each other.

The targeted OaaS scenario requires a workflow engine that serves as a shared platform for several customers. However, each of these customers shall only be aware of his own workflows. Adequate means must assure that the access to is prohibited for foreigners and that a deployed workflow cannot affect workflows of other customers.

RQ12: Scalability

In both, business and scientific scenarios, a workflow engine has to manage the deployment of workflows but also the execution of many workflow instances. In particular, if the workflow engine is used in the OaaS scenario, it has to manage various workflows from various domains representing a multitude of execution behaviors. The workflow engine must be capable of scaling with the number of workflows and must support load balancing abilities.

3.3.2 Workflow Management

Workflow Management comprises the deployment and undeployment of workflows. The following requirements target the component that deals with these activities.

RQ13: Hot-deployment

Workflow Management operations such as deployment and undeployment should not affect running workflows. Thus, the workflow engine needs an hot deployment mechanism that allows the deployment and undeployment without restarting any component.

RQ14: Functional Correctness Tests

The demands sketched in Emmerich et. al [39] imply testing functional correctness of invoked services before deploying a workflow like automated UNIT tests. As this requirement does not directly affect workflow execution, it is a shall-criterion only.

RQ15: Workflow Execution Statistics

Traditional business workflow management systems collect data about finished and currently executing workflow instances for enabling business activity monitoring (BAM). This is an important task of the business workflow life cycle and helps for analyzing the effectiveness of the process, supporting business decisions, or detecting problems during the added value chain. Therefore, it is required to enable the collection of process information not only for a single process but also for deeper analysis of information retrieved from global knowledge.

3.3.3 Workflow Execution

The execution of workflows is the primary function of each workflow management system. These requirements shall be considered for the design of the workflow execution component.

RQ16: Web Service Interfaces

A workflow must provide a Web service interface that offers the full range of functionality. This will allow for handling workflows as a fully integrated part of a SOA. Furthermore, this will simplify the hierarchical composability (see RQ7) of workflows since higher level workflows can use standard invocation mechanisms to invoke a workflow as part of a more abstract process.

RQ17: Variables

To integrate and orchestrate information systems, the workflow engine must be capable of temporarily storing some data for reusability in a later message exchange . This allows for passing message content from one service to another. Furthermore, simple processing of information with means of WS-BPEL is possible, for instance the aggregation of data from several services. Even more important is the ability for using such data for control-flow steering.

RQ18: Fault/Failure Handling

During workflow execution, we have to deal with two kinds of faults. On the one hand, business faults such as the unavailability of a flight in an holiday booking process including hotel, flight, and rental car occur. On the other hand, we have to deal with technical failures, such as a temporarily unavailable service.

Business faults need adequate compensation mechanism to roll back previously processed activities. Normally, these activities are explicitly modeled in the business process description as failure cases. However, a bad response of an invoked service is not limited to business processes. In scientific workflows, such "business" faults can be the unavailability of data or a negative response from a service that automatically checks the correctness of intermediate results. This possibly also requires the compensation of previously changed data. In contrast to technical faults, business faults are reported as SOAP messages.

Technical failures – such as a missing Internet connection or service unavailability – cannot be compensated through workflow logic. Policies like retrying or automatically switching to an equivalent service can enable some failure tolerance but not all errors are temporary or can be masked. If such policies do not work properly, the workflow will fail

due to a non-executable activity. In this case, at least the information at what activity the failure occurred must be made available for the workflow customer as debugging data.

RQ19: Workflow Monitoring

Users must be enabled to monitor the progress of the overall workflow as well as the execution state of each workflow activity. This is important for long running workflows, where the user has to check intermediate results in eScience. Furthermore, this information can be used for debugging erroneous processes when testing the workflow or to detect technical faults during workflow execution.

RQ20: Persistence

All relevant workflow execution data shall be stored persistently. This shall enable the recovery of completed workflows or at least the retrieval of the latest workflow state before the workflow engine crashed.

RQ21: Human Interaction

Human interaction is extremely important in business processes. Often such processes are only semi-automated and need some human input. The workflow framework should provide meachanisms allowing humans to interact with the workflow. Typical human interaction activities are the request for information as the current workflow progress or to make a decision that influences the workflow's actual executed control-flow. For example according to the above mentioned travel agency scenario, a employee has to check the automatically chosen flights before booking them. Depending on his answer the workflow books the flights or repeats the selection process.

RQ22: Quality of Service Properties

When workflow customers are charged for the usage of a workflow, a certain service quality level must be guaranteed. This includes both functional and non-functional properties like reliability, availability, or response time. This requirement is closely related to some other requirements like the handling of technical faults (RQ18), the persistence of process information (RQ20), and scalability (RQ12).

The realization of quality of service (QoS) guarantees is challenging. On the one hand it is complex to calculate an assessment of QoS of a workflow that is potentially composed of dozens of services including human interaction as well as services that are hosted by third-party enterprises or other administrative domains (e.g. Grid sites). In addition, the control-flow of a workflows can be complex as including several compensation activities for different faults or it is highly dynamically if it depends on the results of preliminary executed activities. On the other hand, it is impossible to establish a judicial applicable monitoring system that can determine what service provider or whether the workflow is

responsible for the failure in such a spacious environment. This situation gets even more worse if customers integrate arbitrary or at least self provided services in their workflows.

RQ23: Accounting and Billing

An adequate accounting and billing system is mandatory for a commercially offered for workflow execution. Often accounting and billing data is based on monitoring information: "Who executed which workflow?" and "When was the workflow executed?" are examples for accounting questions. The workflow system should provide mechanisms for collecting this information. The pricing model for the executed workflow depends on the contract between the provider and the customer.

RQ24: Adaptivity

As described, some scientific workflows show experimental behavior. Scientists often process their experiments in a trial-and-error way, needing the possibility to adjust the workflow during execution (ad-hoc changes). This requires direct access of the modeling tool to the workflow engine. [113].

Moreover, Grid workflows usually require the processing of large amounts of data in several steps. It should be possible to specify rules for resource allocation, so that some tasks are executed on the same infrastructure where the data is stored to avoid unnecessary time-consuming file transfers of intermediate results.

RQ25: Provenance

For scientists provenance of data is very important. Experiments have to be repeatable, even in a dynamic environment like a SOA. Sonntag et. al [113] state that business workflow management systems collect information on workflow instance level for an audit trail, but this does not suffice for scientific scenarios. The workflow shall log exact information about search, selection, and integration of all involved services.

3.3.4 Requirements Summary

Requirement	Summary
Basics	
RQ1: Standards	Reuse of commonly accepted standards, such as SOAP and WS-BPEL.
RQ2: SOAP communication protocol	The usage of SOAP as default communication protocol for message exchange.
RQ3: Basic Activities Support:	The workflow language and engine must support all necessary activities for workflow enactment.

Requirement	Summary
RQ4: Role-based Access Control:	The workflow engine must consider user roles for access control.
RQ5: Secure Communication:	The workflow engine must guarantee security during message exchanges, e.g. Transport Layer Security.
RQ6: Integration of new service providers:	It should support a simple integration of new service providers.
RQ7: Hierarchical Composability	Workflows must be callable again from higher level workflows.
RQ8: Stateful Services Orchestration	The handling of stateful services must be supported.
RQ9: WSRF-compliance	The workflow must provide itself as WSRF-compliant service.
RQ10: Control- and data-flow modeling	The workflow description language must be capable of modeling the control and data-flow.
RQ11: Workflow isolation	The system must be realized as a multi-tenancy architecture.
RQ12: Scalability	In the context of OaaS, the system shall be scalable for several hundred customers.
Workflow Management	
RQ13: Hot-deployment	The deployment and undeployment of workflows without restarting any component.
RQ14: Functional Correctness Tests	It shall be possible to test invoked services for functional correctness before deployment.
RQ15: Workflow Execution Statistics	The system shall provide some statistics about workflow execution for further analysis.
Workflow Execution	
RQ16: Web Service Interfaces	The workflow must provide its interface by Web service means.
RQ17: Variables	It must be possible to define variables and steer the control-flow with its content.
RQ18: Failure/Fault Handling	Failures and Faults must automatically be corrected or coped as far as possible.
RQ19: Workflow Monitoring	Each workflow must provide some information about the workflow progress and its state.
RQ20: Persistence	The state of the system must be stored persistently to allow a recovery after a crash.

Requirement	Summary
RQ21: Human Interaction	The assignment of tasks to humans during workflow execution is very helpful for business processes.
RQ22 Quality of Service Properties:	It shall be possible to offer the workflows with a certain degree of QoS.
RQ23 Accounting and Billing:	The workflow engine shall collect and provide accounting and billing information.
RQ24 Adaptivity:	It shall be possible to adapt the workflow at runtime.
RQ25 Provenance:	The workflow shall provide provenance information after execution.

TABLE 3.1: Review of the architecture with respect to the requirements

4 Orchestration Architecture

Contents

The integration of various service types in a secure and flexible way offers advantages for both application domains: business and science. Scientists are enabled to combine the complex Grid service infrastructures with free, commercial, and self-provided services. Enterprises are empowered to extend their business processes by integrating Grid and Cloud services to cooperate with partners or to offer value added workflows in the context of the Orchestration as a Service paradigm. In return, OaaS is also interesting for scientists since it allows for providing standard experiments encoded as workflows to junior scientists or students or for provenance reasons.

The realization of such an integration infrastructure must meet many requirements. This chapter describes the architecture for the secure orchestration of Grid and Cloud services; a workflow engine that enables workflow execution across several administrative domains as it integrates services that require different security mechanisms. The general concepts of the architecture are illustrated by reference to the UNICORE 6 Grid middleware system but are also applicable to other service frameworks.

The chapter is structured as follows: Firstly, we discuss the selection of base technologies and standards to realize the architecture. In this process, we consider the requirements from Section 3.3. The major goal is to integrate various kinds of services into one workflow. Section 4.1 presents the overview of the architecture, resulting problems, and our solutions to these problems.

Secondly, in Section 4.2 outlines a security infrastructure that is applicable to the idea of Orchestration as a Service. It must be possible to integrate several existing identity management systems from companies or scientific institutions to a low a concurrent usage. Additionally, access control must be implemented in a fine-grained and role-based manner.

Section 4.3 addresses the integration of external services into workflows. On the one hand, this includes the handling of stateful WSRF-compliant services in a WS-BPEL workflow and on the other hand a plug-in technology to support various security mechanisms for the invocation of external services.

The chapter concludes with a short overview on how to integrate human interaction into workflows without extending the orchestration language.

4.1 Architecture

The first important decision is not to create a new workflow engine from the scratch but to build upon an existing and well tested business process management system. Today, open-source business process management systems are available that are well tested and already used in a productive manner. Furthermore, existing solutions from this domain already fit into the business process life cycle. The architecture will extend a workflow engine to meet the requirements for cooperations between enterprises. Since the engine originates from the business domain, this section outlines how to integrate Grid and Cloud services in business workflows and evaluate the resulting engine with respect to its application to scientific workflow in Section 6.

In order to integrate various Web service technologies with a special focus on different security frameworks, an extendable architecture is required. The engine must be build upon well known and established standards for orchestration and security to reach a high user acceptance. We discuss these basic standards and technology frameworks in Section 4.1.1.

The subsequent subsections present the final architecture. It mainly consists of two service extensions for workflow management and workflow execution that we implemented as stateful services into a Grid middleware. Furthermore, we present some ideas about load balancing and fault handling in this architecture.

4.1.1 Technology Selection

The selection of appropriate basic technologies and standards is the first step in the design process towards the new workflow engine. The chosen technologies should already provide as many functions as possible so that we can focus on new requirements like security and the integration of different service types.

As basic technologies, we firstly require a suitable workflow description language, secondly a basic Web service framework to implement the service extensions, and thirdly an appropriate workflow framework to interpret this language.

Orchestration Language

The capabilities of the workflow description language essentially influence the capabilities of the resulting workflow engine. For choosing the language, we consider language related requirements such as RQ2, RQ3. In the following, we discuss different workflow description languages from both application domains and present the final candidate.

During the past years, the Grid domain created several workflow execution languages since different scientific communities developed them in parallel. These languages are often designed for one specific community or application scenario. Thus, there is a multitude of scientific workflow languages and engines today and no commonly accepted scientific orchestration standard. This fact precludes the selection of the most accepted Grid orchestration language.

For instance, GWorkflowDL [4] and the Grid Flow Description Language [54] are both languages based on Petri Nets. Both are capable of orchestrating Globus Toolkit Grid services. They provide different modeling tools for the respective workflow engine and language. None of the engines provide Web service interfaces to communicate with the workflow as demanded by RQ16. This is typical for pure Grid service orchestration engines since Grid workflows are often started directly from the workflow editor. In such cases, the input data for the workflow is provided by a data source that is directly addressed in the workflow.

Although most Grid workflow engines do not offer WSDL interfaces, they use SOAP for communication, because modern Grid services are build upon SOAP-based Web service frameworks. An integration with arbitrary services is not possible and beyond the scope of the design goals. Additionally, the capability to manipulate the control-flow based on workflow data (see RQ17) is missing in most Grid workflow languages.

Furthermore, limited failure handling and roll-back mechanisms (RQ18) and e.g. the missing integration of human interaction (RQ21) are arguments against the usage of a Grid workflow language as basis for the Grid and Cloud services enabled workflow engine.

Thus, GWorkflowDL and most other Grid workflow description languages are unsuitable for the technical support of business processes because many requirements such as RQ1, RQ3, RQ6, and RQ7, are not sufficiently solved. For further details about Grid workflow systems see Section 7.2.

As described in Section 2.2.1, WS-BPEL is the de-facto standard for workflow orchestration in the business domain. Therefore, it is a candidate for an orchestration engine that enables advanced workflows in cooperation scenarios in the business domain.

Since the workflow engine shall be applicable for both domains, we have a closer look on the applicability of WS-BPEL for the scientific domain: The general appropriateness of BPEL4WS [7] (the predecessor of WS-BPEL) and WS-BPEL for scientific workflow execution has already been discussed in several publications. Leymann [89] states that BPEL4WS fulfills many requirements to orchestrate WSRF-based Grid services. Dörnemann et al. [38] describes how the WS-BPEL extension mechanism can be used to introduce new BPEL activities that enable Grid service communication. Both extended a workflow engine with activities for the creation of a WSRF instance (Grid create), the invocation of this WSRF instance (Grid invoke), and the destruction of the instance (Grid destroy). Slomiski [111] describes ideas about how to extend BPEL4WS for the integration with OGSI [124] and WSRF services. He also uses the BPEL built-in extensibility mechanisms. Emmerich et al. [39] illustrate their experience in orchestrating scientific workflows gained from an extensive case study for the automation of a polymorph prediction application, using BPEL4WS. They describe the extent with which BPEL4WS supports the definition of scientific workflows. Furthermore, they present the evaluation of reliability, performance, and scalability of the open source workflow engine ActiveBPEL on executing this complex scientific workflow.

The applicability of WS-BPEL for scientific workflows is closely related to the topic of the applicability of business workflow management systems for scientific workflows. In [113], Sonntag et al. state that such systems are limitedly appropriate for the orchestration of scientific workflows. They conclude with a list of seven points that should be improved but they do not doubt the applicability of WS-BPEL for scientific workflows. Other publications [40, 112] also agree that WS-BPEL is generally capable of orchestrating eScience workflows.

In [56], we discuss the appropriateness of WS-BPEL for Grid service orchestration according to the identified requirements. In the following, we outline the advantages of WS-BPEL with respect to the extended requirement analysis presented in Section 3.3:

- RQ2: WS-BPEL is designed as a workflow engine to orchestrate Web services that communicate with SOAP messages.

- RQ3: The WS-BPEL specification only comprises the communication with SOAP-based Web services. Security issues and the orchestration of stateful services is beyond the goals of the specification. Most Grid and Cloud services are based on standard SOAP communication and mainly differ technologically in security requirements. Hence, we assume WS-BPEL is generally applicable to invoke Grid and Cloud services if the corresponding security mechanisms are supported (cp RQ6).

- RQ7: The standard defines that each workflow advertises itself as a Web service using WSDL for interface description. This enables the hierarchical composition of workflows as workflows can be invoked from higher level workflows.

- RQ10: WS-BPEL allows the definition of the control-flow e.g. by using conditions, loops, sequences, or flows. Data-flow modeling, as it is necessary for scientific workflows is more difficult, since the WS-BPEL is designed to be interpreted in a central orchestration engine. Small data amount can be passed through the engine as SOAP messages and can also be used to affect the control-flow. But WS-BPEL is not capable of processing large amounts of data or data streams since there is no activity for a peer-to-peer data transfer between two invoked services. Furthermore, SOAP is extremely inefficient as a transport protocol for large data amounts.

- RQ16: Each workflow features a Web service interface with a valid WSDL description. The workflow is accessible via SOAP message exchange pattern.

- RQ17: A WS-BPEL process stores all messages in variables and offers the possibility to define arbitrary variables that can store combined or processed data. The XPath standard [31] standard enables the selection of single information and the application of arithmetical, logical, and compares operators. This allows the formulation of complex conditions to alter the control-flow.

- RQ18: WS-BPEL is able to react on business (or logical) faults if the designer models the workflow in an appropriate fashion. Technical faults such as communication errors are beyond the fault handling capabilities of the WS-BPEL process description.

To conclude, we state that WS-BPEL is generally applicable for modeling and executing workflows that are composed of SOAP-based Grid and Cloud services.

Basic Web Service Framework

The second basic technology is an appropriate Web service framework for the implementation of additional functions. This selection has a major impact on the final design of the architecture. The workflow engine will combine Web services, Grid services, and Cloud services. It is obvious to reuse a technology stack from one of these domains. Cloud computing is mainly commercially driven and each provider uses a different technology stack. Hence, we limit the selection process on freely available and open source service frameworks for Web services, Grid middlewares, and the possibility of extending an existing WS-BPEL workflow engine.

Grid middleware frameworks are based on Web service frameworks with additional functions as WSRF-compliance and advanced security. According to RQ9, pure Web service frameworks are not the first choice. Thus, we choose between

- a Grid middleware that does not provide a WS-BPEL workflow engine implementation or
- a WS-BPEL engine and the corresponding Web service framework. Such an engine normally neither supports Grid security standards nor stateful services.

RQ1 states that the architecture should be built upon standards to increase the acceptance, especially in the business domain. Therefore, we have to consider the costs of extending one of the above listed components with the additional required standards. The reimplementation of the WS-BPEL standard into a Grid middleware is beyond question because this would be much too labor-intensive and error-prone and no simple library for orchestration is available.

The extension of a WS-BPEL engine seems to be more promising because several open source WS-BPEL engine like ActiveBPEL and ApacheODE are available and offer explicit extension points. However, the extension would require at least the following tasks:

- Firstly, we have to enable the handling of stateful and WSRF-compliant Grid services.
- Secondly, we have to implement advanced security and configuration mechanisms for invoking Grid and Cloud services.
- Thirdly, we have to secure the WS-BPEL engine with Grid security standards and enable role-based access control. Furthermore, this security architecture must be capable of serving as a multi-tenant platform.

These changes would require wide modifications on the WS-BPEL engine, as the following example shows. Dörnemann et al. [38] patched an early version of ActiveBPEL workflow engine (version 2.1) to invoke Grid Services by introducing a new WS-BPEL dialect. They implemented new Grid invoke activities for the creation, the usage, and the destruction of WSRF instances. However, the creation of such an proprietary WS-BPEL dialect results in a proprietary workflow engine implementation since the new language is only supported by this modified workflow engine. This heavily decreases the sustainability and acceptance of the solution. If the WS-BPEL standard evolves, the dialect must be adapted to the new version. Furthermore, enterprises are not able to keep a perhaps already existing WS-BPEL compliant engine if they want to integrate Grid and Cloud services in their workflows. For these reasons, we agree on neither to modify the WS-BPEL standard nor to implement new features in the WS-BPEL workflow engine. Hence, we decided to use and not to extend an existing WS-BPEL engine implementation. Instead of this, we implement the additional functions as services in a Grid middleware that again communicates with a standard WS-BPEL engine.

The most famous Grid middlewares are based on Web service technologies: UNICORE 6 is build upon the WSRFlite framework and Globus Toolkit 4 (GT4) upon the WS-Core framework. In [84], Kübert et al. examined the performance of these two frameworks. Additionally, they also considered the Apache Muse[1] framework which is also WSRF compliant. They conclude the paper with the statement that "WSRFLite performs dramatically better then both GT4 and Muse. GT4 is in general performing better than Muse" [84]. Additionally, UNICORE 6 already supports some interesting authorization features like *eXtensible Access Control Markup Language* (XACML) [41] and a rudimentary role based access control system. In contrast, GT4 is based upon an old version of the Apache Axis framework and only its security is partly based upon proxy certificates that are not a commonly accepted standard.

As a result of the issues discussed above, UNICORE 6 is the first choice as basis Web service framework candidate for the realization of the architecture. This decision partly meets the following requirements:

- RQ1: UNICORE 6 is based on several well-known and accepted standards such as WSRF, XACML, and personal X.509 certificates.
- RQ2: UNICORE 6 is based on the WSRFlite framework that is a SOAP based Web service framework for stateful services.
- RQ4: The identity management system of UNICORE 6 already includes rudimentary roles. Additionally, the UNICORE VO System (cf. Section 2.3.3) that can be used as sophisticated identity management system is well tested and fully integrated with UNICORE 6.

[1] `http://ws.apache.org/muse/`

- RQ5: Encrypted HTTPS channels are used for the communication with and between the UNICORE 6 components.
- RQ9: WSRFlite is a WSRF compliant Web service framework.
- RQ20: The WSRFlite framework stores a service instance's state persistently.

WS-BPEL Engine

Generally, the solution presented in this section is devised to be independent of the concrete WS-BPEL implementation. However, we have to select a workflow engine for the prototypical realization. Some of the requirements – like monitoring – are only realizable if the WS-BPEL engine offers some additional services beside the actual Web service orchestration.

In [62] and in Section 2.2.2, we presented an analysis of several workflow engines and their main properties. This analysis reveals that fully WS-BPEL compliant and open source workflow engines are rare. As result of the analysis, ActiveBPEL[2] and ApacheODE[3] are the most promising candidates. Finally, we chose ActiveBPEL because the engine is distributed commercially and the vendor[4] offers support for the open-source community.

Beside the ability to execute WS-BPEL workflows, ActiveBPEL offers some additional services that help to meet the following requirements:

- RQ13: The deployment and undeployment can be done by moving and removing deployment archives to/from the deployment directory. Furthermore, ActiveBPEL offers a Web service to deploy workflows. A Web service for undeployment is missing.
- RQ19: ActiveBPEL provides a Web Service interface which allows for requesting a list of all finished workflows and the current execution state of running workflow instances.
- RQ20: ActiveBPEL can be configured to use a database to store the current execution state in a persistent manner. That way, this information is still available after a crash and a process recovery becomes possible.

[2] `http://sourceforge.net/projects/activebpel502/`
[3] `http://ode.apache.org/`
[4] `http://www.activevos.com/index.php`

- RQ23: The monitoring information also includes a detailed process execution log with timestamps. It can serve as basis for accounting data. Hovever, in the default version, it is not possible to map workflows to users since the log does not include authentication data.

4.1.2 Main Component Overview

As stated in the previous section, the new orchestration engine architecture should be composed of the following two components:

1. A WS-BPEL compliant workflow engine that forms the basis for the orchestration process. We choose the ActiveBPEL workflow engine for the prototype.
2. Service extensions to the Grid middleware UNICORE 6 that realize the additional functions as required.

The general architectural idea is the realization of the additional functions as a two way proxy deployed in the Grid middleware. This proxy completely encapsulates the WS-BPEL workflow engine. The additional functions are completely transparent for the WS-BPEL workflow engine. By doing so, we distinguish the workflow processing that is done at the WS-BPEL workflow engine and the additionally implemented functions realized as the new services.

An overview of the architecture is shown in Figure 4.1. The figure focuses on the most important aspects of the architecture and does not comprise all details.

The shown WSRF-compliant services, the *Workflow Management Service* and the *Workflow Service*, are the core service extensions for workflow management and workflow execution. Together, these services and the arbitrary WS-BPEL engine, in this case ActiveBPEL, represent the **Grid and Cloud workflow engine**. For each workflow – deployed via an instance of the Workflow Management Service – one version of the Workflow Service will be created using a hot deployment mechanism (RQ13) without restarting. Each Workflow Service represents the functions of exactly one deployed workflow. Each running workflow is represented as Workflow Service instance according to the WSRF standard (RQ9).

The prototype realizes the service as Grid Services within the WSRFlite service container hosted within the UNICORE 6 runtime environment. These service extensions are placed next to the default UNICORE 6 services but they have no direct dependencies. Of course, invocations of arbitrary UNICORE 6 services can be done in workflows.

Figure 4.1 also highlights that the WS-BPEL workflow engine is only loosely-coupled with the additional services via HTTP(S) communication channels. This allows a separate

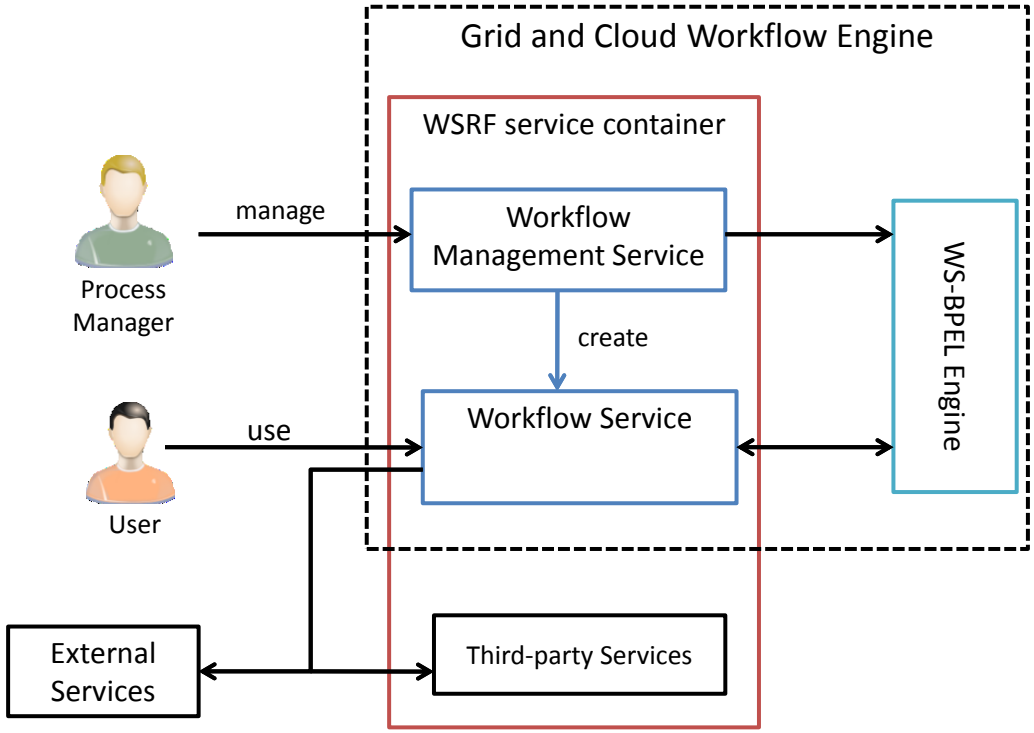


FIGURE 4.1: Components of the workflow engine

deployment of the WS-BPEL workflow engine on extra hardware and the usage of several WS-BPEL workflow engines with one additional services installation – for example for load balancing issues (cf. Section 4.1.6).

Despite all these advantages, some problems arise when using such a decoupled architecture without WS-BPEL extensions:

Firstly, the WS-BPEL code that is necessary to call a Grid service is slightly more complex than the introduction of new Grid-specific activities as presented in [38] since we have to explicitly deal with the invoked services' instance IDs. A Grid service requires to explicitly create and destroy the service instance. At least three WS-BPEL invoke activities are necessary for the invocation of one stateful service. We tackle this problem by hiding the complexity from the user and also as far as possible from the workflow designer. Therefore, we present some basic Grid service usage pattern that can be used as template for these invocations. A sophisticated BPEL editor could present this pattern in a much simpler way e.g. as a single Grid invoke activity symbol. The pattern is described in more detail in Section 4.3.1.

Secondly, the realization of workflow management and the mapping between the Workflow Service and the service that represents the workflow in the WS-BPEL workflow engine differs from engine to engine and requires some additional effort for the adaption to new WS-BPEL engines (see Section 5.3).

Thirdly, we have to deal with two instances of the same workflow, one instance in the Workflow Service and one instance in the WS-BPEL workflow engine. The mapping of these two instances on each other also requires additional effort (cf. Section 4.1.5). Furthermore, the retrieval of the current execution state of the WS-BPEL workflow instance depends on the capabilities offered by the WS-BPEL engine because they are beyond the scope of the standard. Here, we also need an adaptable solution to deal with different WS-BPEL workflow engines.

4.1.3 Workflow Management Service

The *Workflow Management Service* is a stateful service for workflow deployment and workflow management. It consists of a factory and the actual service implementation, both services with different WSDL interfaces. The function range comprises operations for workflow deployment, undeployment, redeployment, and a search operation for deployed workflows. The use cases are depicted in Figure 4.2. We assume the user to be an expert who has detailed knowledge about workflow modeling and deployment requirements.

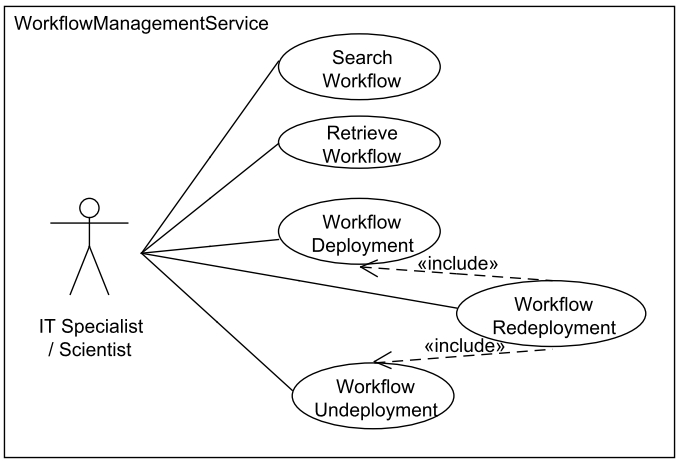


FIGURE 4.2: Use cases – Workflow Management Service

The redeployment is an immediate undeployment of the workflow description and a subsequent deployment of the new workflow. Therefore, the redeployment includes the deployment and undeployment use cases that are published in [58] and also presented in this section.

The deployment starts after creating a new Workflow Management Service instance and sending a deployment package (see Section 5.4) to this service instance. The deployment package comprises all necessary deployment information such as the BPEL code, the interface descriptions (partners and workflow) as WSDLs, and the deployment descriptor.

Controlling the access to the Workflow Management Service instance is crucial since the detailed process information and therewith internal business details are published. Therefore, each instance has an owner who is identified by his certificate's distinguished name. The default owner is the person who created the instance. The default access policy states that only the owner and administrators are allowed to access the Workflow Management Service instance, but more fine-grained policies are possible. Additionally, the instance stores more detailed information about the creator such as the affiliation, department, role, or similar attributes included in the creator's identity. With minimal implementation effort, this information can be made a part of the authorization process as well.

Beside the default creation method, the factory service provides the possibility to search for deployed workflow descriptions. As search index the workflow name and as search parameter regular expression are used. If the the architecture is applied to an OaaS sce-

nario, it is necessary to filter the results according to the invokers's identity to prevent the workflows of other tenants from being included in the answer (RQ11). The filter therefore considers the enriched creator information. The search response message contains a list of Workflow Management Service instance as endpoint references whose names match with the search parameter and the filter rules.

Since the Workflow Management Service is a WSRF service it publishes its state information via the default WSRF operations [14]. In the current prototype, all state information of a Workflow Management Service instance is immutable. Only by triggering the redeployment operation (or a consecutive undeployment and deployment), the state information changes. The instance state comprises the packed deployment package but also detailed information such as the deployment descriptor or the default access policies for the Workflow Service and the corresponding factory.

During deployment, a new Workflow Service is created and the endpoint reference to the Workflow Service Factory is published as state information of the Workflow Management Service instance. This Workflow Service Factory offers means to create new instances of a Workflow Service that represent exactly one instance of the executable workflow (cf. Section 4.1.4).

The following enumerations provide a more detailed view on the two main use cases, the deployment and undeployment of workflows (see [58]). These processes can be subdivided into several steps. If one step fails the complete deployment or undeployment process fails, and the preceding steps must be rolled back. The deployment process is as follows (compensation steps are omitted for the sake of clarity):

1. The workflow deployment package is stored in a previously specified local file space and is unpacked.
2. The deployment package is checked for correctness and completeness as far as possible.
3. The WS-BPEL workflow description and WSDLs are modified to solve the WS-BPEL/WSRF workflow engine mapping problem. Details about this problem and the solution are described in Section 4.1.5.
4. The workflow is deployed to the WS-BPEL workflow engine. An adapter concept allows a deployment process that automatically adapts to the used WS-BPEL workflow engine implementation.
5. The corresponding Workflow Service (see Section 4.1.4) is created and registered with the service container.

Subsequently, undeployment is executed as follows:

1. The factory service of the corresponding Workflow Service (see Section 4.1.4) is deregistered and removed from the service container to prevent the creation of new Workflow Service instances.

2. The undeployment process waits for the termination of active workflow instances. The initiator of undeployment may decide whether these instances shall terminate normally (expiration date is infinite), instantly (expiration date is *0*), or at a specific date (expiration date is specified either explicitly or by a grace time). Except for normal termination, Workflow Service instance termination is enforced by the undeployment process at the given expiration date. By default, the normal termination strategy is used.

3. The actual Workflow Service (see Section 4.1.4) is deregistered and removed from the WSRF service container.

4. The WS-BPEL workflow is undeployed from the WS-BPEL workflow engine by using the above mentioned adapter. It has to be ensured that all data concerning the WS-BPEL workflow to be undeployed is removed.

5. The workflow deployment package and all related data are removed from the respective local file space.

Since the Workflow Management Service is WSRF-compliant, the instance has a limited lifetime, but the user can extend this lifetime regularly or even set it to unlimited to prevent an unintended deletion of the Workflow Management Service that includes the immediate undeployment of the workflow and all associated services and data.

4.1.4 Workflow Service

The *Workflow Service* is a generic two-way proxy for workflow enactment deployed in the service container. As it is WSRF-compliant, it offers all state information as resource properties accessible via the default WS-Resource [53] operations. For workflow instance creation, the service provides an additional service factory, the Workflow Service Factory.

Figure 4.3 illustrates the use cases for the Workflow Service. The main actor is the workflow user. Examples for possible other users are scientists who process in-silica experiments or employees who are part of a business process. These workflow users are associated with the workflow enactment use cases such as the creation of workflow service instances, the actual workflow execution, the retrieval of workflow state information for monitoring, and the alteration of the workflow configuration. The prototype supports the reconfiguration of external service invocations including the redirection of messages to

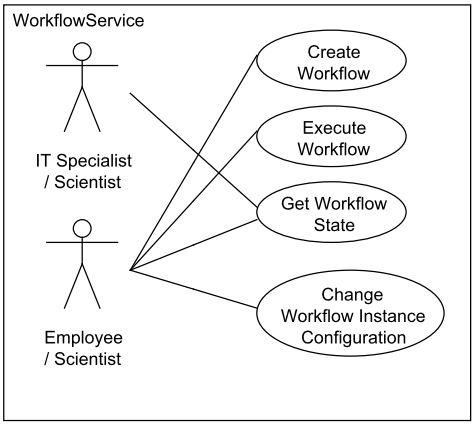


FIGURE 4.3: Use cases – Workflow Service

equivalent services or the adjustment of security configurations for a successful communication.

Since workflow users are often no technical experts, simple user interfaces and appropriate client applications should be provided for facilitating the use cases. Such applications should hide the complexity of WSRF-compliant services for example by implicitly creating workflow instances when a new workflow starts. The users must not necessarily know about the internal technical processes.

The second participant in the Workflow Service use cases is an IT specialist. He collects monitoring information about currently and finished running workflows to get an overview about the ongoing business processes. This enables the complex process analysis by the aggregation of state information from several workflow instances.

During the deployment process at the Workflow Management Service instance, a specialized version of the generic Workflow Service is registered at the service container that represents this new workflow. The interface of the workflow service is composed of three parts:

- The *workflow execution operations* that are also offered by the workflow's WSDL at the WS-BPEL workflow engine.
- The default *WS-Resource operations* to retrieve and edit the state of the service instance.
- *Additional operations* to provide additional information to the workflow for example to deposit credentials for credential delegation.

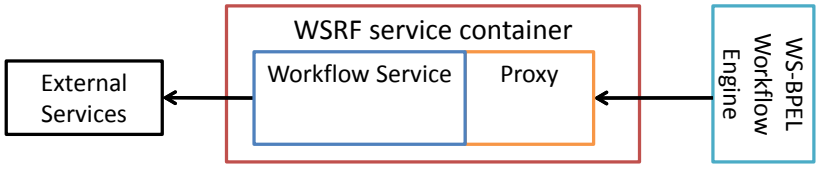


FIGURE 4.4: Integration of the proxy

Since all operations are combined in one interface, a client sends messages generally to the same service. The service is responsible for the correct handling of the message. If a *workflow execution* operation is triggered it forwards the message to the WS-BPEL workflow engine. But before this, it removes all Grid or Cloud specific tokens from the message so that such details are transparent for the WS-BPEL workflow engine. Messages dedicated to the other operations are processed in the Workflow Service.

The other way around, the existence of the Workflow Service should also be transparent for the WS-BPEL workflow engine. Therefore, it is necessary to reconfigure the hosting environment of the engine (Tomcat in case of ActiveBPEL) to use a HTTP(S) proxy. All messages that are directed to external services are rerouted to the proxy. The WSRF service container provides an additional proxy component that is capable of receiving the messages and dispatching them to the correct Workflow Service instance (see Figure 4.4). At the instance, the message is modified according to the configuration and then forwarded to the external service. Synchronous answers are returned the same way backwards. Asynchronous answers are handled as described above. The usage of a HTTP proxy is a common feature that is supported by nearly all service containers and will not limit the exchangeability of the WS-BPEL workflow engine.

However, not all HTTP calls are forwarded to a Workflow Service instance. For instance, during workflow deployment, the WS-BPEL workflow engine requests additional resources as referenced WSDLs or XML Schema files if these are not included in the deployment package. The proxy distinguishes such messages from workflow enactment messages and acts as a standard HTTP(S) proxy in these cases: It simply forwards the request to the external service and passes the response back to the WS-BPEL workflow engine.

4.1.5 WS-BPEL/WSRF instance mapping

The loosely-coupling of the WS-BPEL workflow engine and the Workflow Service instances which add additional functionality for Grid and Cloud service integration requires the synchronization of information between both components. Figure 4.5 depicts a typical

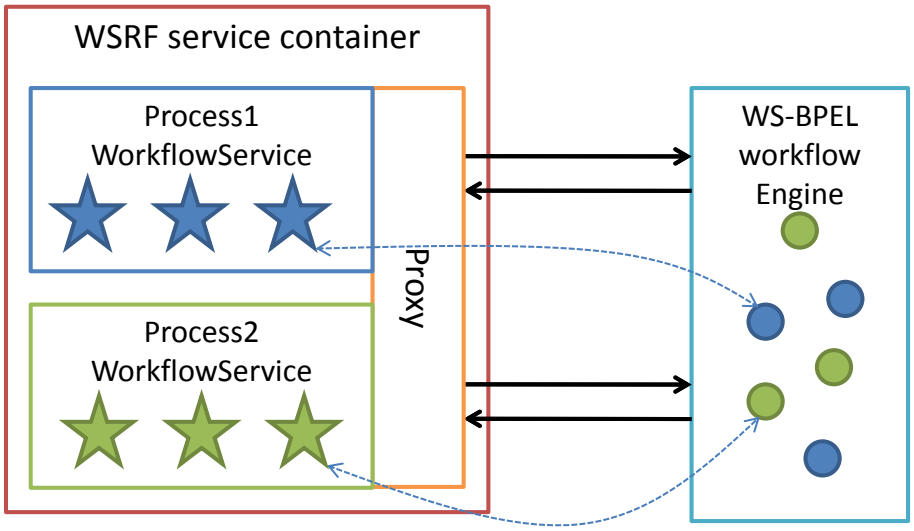


FIGURE 4.5: Mapping of Workflow Service and WS-BPEL engine instances

situation with two deployed workflows and therewith two Workflow Services. Each currently running workflow exists twice, in the WS-BPEL workflow engine (circles) and the respective Workflow Service instances (stars) at the service container. Thus it is necessary to map these instances to each other to allow an unique assignment of messages to the corresponding instances.

The mapping requires the exchange of IDs that uniquely identify the instances at the partner component. This is done by passing the identifiers to the opponent as additional payload during the normal message exchange. Since we do not want to adjust the WS-BPEL workflow engine, we decided to handle the IDs as part of the WS-BPEL workflow. This requires modifications in the WSDL files to add the ID as message payload as well as to apply changes in the WS-BPEL code to store the ID and reinsert it in outgoing messages during workflow execution. We address these aspects by applying *architecture-specific patterns* [24]. These patterns are automatically applied to the WS-BPEL process description during process deployment via XSL transformations. As a result, the existence of the patterns and the entire mapping problem is completely transparent to workflow users and designers.

On the one hand, the identification of the Workflow Service instance at the WSRF-compliant service container requires the identifier to be included in every message initiated by the WS-BPEL engine. For instance, the service instance in UNICORE 6 is identified by a *resource ID* consisting of the service name and the unique instance ID. Instances of a

WS-BPEL process are identified by their *process IDs*. On the other hand, the identifier of the workflow in the WS-BPEL workflow engine is necessary to define the instance if we want to use additional services provided by the workflow engine.

For the first mapping, we applied two patterns to the WS-BPEL code and to the WSDL interface of the workflow: The pattern *On Receive ID Retrieve* expects that each message sent to a process instance provides the resource ID of the corresponding Workflow Service instance. Within the process instance, this ID is stored in a specific WS-BPEL variable. The second pattern, *Pre-Invoke ID Assign*, ensures that the previously stored resource ID is appended to the message header of each message that is sent from the process instance by using an appropriate endpoint reference. A more comprehensive example of these modifications can be found in [68]. The patterns are described in more detail in Table 4.1 as one pattern to solve the instance mapping problem.

The second mapping is of minor importance because the basic orchestration is already possible if only the first mapping is applied but it enables additional functions. To request data about a process instance, the Workflow Service instance needs to know the process ID at the WS-BPEL workflow engine. The handling and even the existence of process IDs is beyond the scope of the WS-BPEL specification. Hence, different WS-BPEL engine implementations use different concepts to address workflow instances. ActiveBPEL, for example, simply increases a process ID counter with every new process instance.

We address this issue by extending the *Pre-Invoke ID Assign* pattern to assign some WS-BPEL process ID to each outgoing message. The pattern is described in Table 4.2. It is abstract enough to handle arbitrary ID data types. Again, a detailed example can be found in the project deliverable [68]. Since the internal ID management of workflow instances is not part of the WS-BPEL specification, the pattern has to be adjusted for each WS-BPEL engine implementation.

4.1.6 Load balancing

The OaaS scenario requires a load balancing concept that allows for the execution of several thousand workflows simultaneously (cf. RQ12). We present some ideas for load balancing which are refined in this section in more detail in [57]. Since the architecture consists of two components, each component can be a bottleneck during workflow execution.

The loosely-coupling of the WSRF service extensions and the WS-BPEL workflow engine offers the possibility to install both components on different hardware. Generally, the workflow processing steps which are executed in a WS-BPEL engine are rather simple – mostly sending or receiving messages or copying small data amounts between variables. The processing steps for a single message in the service extensions are also rather simple

Motivation	In the architecture, messages sent from a process instance must be mapped to the corresponding Workflow Service instance.
Intention	Change the WS-BPEL process description and its WSDL interfaces to manage a unique ID that identifies a corresponding Workflow Service instance. Regarding UNICORE 6, the service name and the unique instance ID can be used for identification.
Structure	*(1) WSDL:* Identify the WSDL representing the interface of the WS-BPEL process and insert an additional message part for each incoming message to carry a WSRF service instance ID. *(2) WS-BPEL:* (a) Create a new process variable that shall store a WSRF service instance ID. (b) Insert an `assign` activity after each `receive` or `pick` activity that has an attribute `createInstance` set to *yes*. This `assign` copies a WSRF service instance ID that is appended to an incoming start message into the previously created process variable. (c) Create a new process variable that shall store a dynamic endpoint reference. (d) Before each `invoke` activity, insert an `assign` activity that initializes a dynamic endpoint reference by copying a literal XML-skeleton representing the type `EndpointReference` into the variable previously created, copying the service instance ID into the `ReferenceProperties` part of the variable, and copying the variable to the *partnerLink* of the Grid Service to be invoked. The BPEL code and the `EndpointReference` literal XML-skeleton is presented in [68].
Participants	The proxy service, the WS-BPEL engine.
Behavior	The Workflow Service instance attaches its ID to each message addressed to the WS-BPEL engine. The targeted WS-BPEL process stores the ID and attaches it to each outgoing message — e. g. for external service invocation. The proxy service then uses the ID to forward the message to the corresponding Workflow Service instance, where the ID is removed from the message header before it is redirected to the external service.
Consequences	(1) Changes of both the WSDL description and the process description are necessary. (2) Non-domain-specific code is inserted into the WS-BPEL process description (i. e. code not being part of the domain-specific service to be represented by the process).

TABLE 4.1: Pattern WSRF/WS-BPEL Instance Mapping

Motivation	Workflow management and monitoring require the identification of running WS-BPEL process instances.
Intention	Implement a mechanism that propagates the ID of WS-BPEL process instances to the Workflow Service instance.
Structure	Extend the pattern *Pre-Invoke ID Assign* by modifying the `assign` activity that is inserted before each `invoke` activity in the respective WS-BPEL description (cp. Table 4.1): (1) Extend the `EndpointReference` XML-fragment to store a process ID. (2) Add a `copy` statement that uses the WS-BPEL engine's function to retrieve the process ID and that stores it in the newly created XML-fragment part.
Participants	The Workflow Service, the WS-BPEL workflow engine.
Consequences	(1) Changes of the WSDL description are unnecessary because the SOAP header is used to propagate the process ID. (2) The Workflow Service is able to address the WS-BPEL process instance via its ID only after the first `invoke` activity has been executed. (3) Non-domain-specific code is inserted into the WS-BPEL process description.

TABLE 4.2: Pattern Process ID Retrieval

because most messages are forwarded with only small modifications. Even the security checks are not very costly for a single message. However, the mass of messages, message transformations, and workflow executions for example in an OaaS scenario can increase load on the components and possibly cause a bottleneck. This section describes some ideas to deal with bottlenecks but these ideas are not evaluated in detail.

The design of the Workflow Management Service and Workflow Service allows the management of several WS-BPEL workflow engines on different resources. When a new Workflow Service instance is created, the factory assigns one of the available WS-BPEL workflow engines to the new instance according to a load balancing strategy (e.g. Round Robin). This will facilitate scalability if the WS-BPEL workflow engine is the bottleneck.

If the Workflow Service itself is the bottleneck load balancing is slightly more complicated because UNICORE 6 is not shipped with load balancing for services. Since the services are stateful, we cannot use a standard Web service load balancer that dispatch messages without considering dependencies between two messages directed to the same instance. The following enumeration presents some load balancing strategies for the Workflow Service with a special focus on the WSRF-properties and with respect to the OaaS scenario:

1. *Customer-based balancing:* It is possible to install one instance of the secure Grid and Cloud workflow engine for each customer. Then, the dispatching mechanism

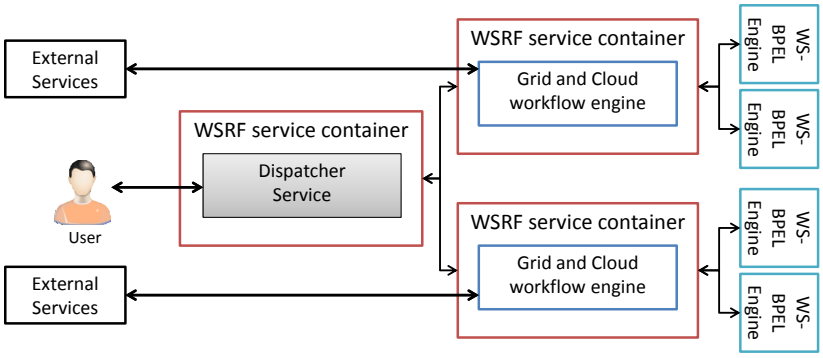


FIGURE 4.6: Load Balancing Architecture

of the UNICORE Gateway can be used for load balancing. If a server is overloaded, the customer rents another server that is registered with the gateway. The customer handles the mapping of workflows and workflow instances to the workflow engine instances on his own. This solution is not very sophisticated since a lot of resources on less used servers are wasted and the system does not scale in the sense of Cloud computing.

2. *Workflow-based balancing:* A dispatcher that tries to balance workflows according to their load on several installations can improve load balancing. The balancer represents all interfaces to the user and dispatches the messages according to the workflow name. This allows the coexistence of workflows of different customers on the same machine. It will still be problematic if one workflow exceeds the capacity of one Grid and Cloud workflow engine.

3. *Workflow-and-resource-based balancing*: If a workflow gets more popular and load increases, a single machine might not suffice. The system can deploy the workflow on another resource and dispatches the messages not only according to the workflow name but also considering the resource ID. The Workflow Service Factory must register new service instances at the dispatcher service so that it can route the messages correctly. In [57], we presented a slightly different version of this strategy wherein all workflows are deployed on all resources.

Figure 4.6 shows an overview of a load balanced landscape of two cooperating Grid and Cloud workflow engines. Each engine splits up the workflow instances to altogether four WS-BPEL workflow engines. The already mentioned dispatcher service forwards the messages to the installations according to one of the above presented strategies.

4.1.7 Fault handling

Errors can occur in various shapes during the execution of a WS-BPEL process. We discuss the three main fault scenarios and how the Grid and Cloud workflow engine can handle these (cf. RQ18).

Firstly, faults can affect the business logic. For example a travel agency scenario: if a hotel and rental car is available but the matching flight is overbooked, the workflow must cancel the rental car and hotel bookings. We call such faults business faults. They must be handled on the business and not on the technical level. WS-BPEL offers means – such as compensation and event handler – to deal with business faults. Such handlers allows for rolling back already processed workflow tasks by additional service invocations or retrying some tasks with different parameters.

Secondly, external services can fail. This is critical because a connection exception generally causes the complete workflow to fail. Generally, a workflow engine provider has no influence on the availability of the involved services since they are not under his administrative control. A common method to coat such errors is to use some kinds of message buffer that resends the message after a time period hoping that the service is available again. In addition, standards like WS-Reliable Messaging [44] can help to increase fault tolerance. However, if the service will not be available for a longer time period, even such mechanisms must report an exception to the workflow. The handling of such technical errors is not in the scope of the WS-BPEL standard. Since the architecture uses SOAP message communication the available coating mechanisms can be reused to increase fault tolerance.

Thirdly, the architecture engine includes an additional potential point of failure. If one of the two components – WS-BPEL workflow engine or the service extensions – fails the workflow will fail, too. If the service extension crashes, the WS-BPEL engine encounters a connection error and the corresponding workflow fails. Clients also receive errors that the Grid and Cloud workflow engine is unreachable but they can try to resend the message later. If the WS-BPEL workflow engine fails the service extension returns faults to the callers. Both components are able to store their state persistently, but, at the moment, the components do not synchronize state and recovery information with the partner.

In the current version, technical fault handling is not implemented in the prototype but business faults can be handled by means of WS-BPEL.

4.2 Workflow Security

Security is one of the most important issues when executing workflows across multiple enterprises' boundaries and especially when using the Internet for message transport. The thesis considers different kinds of security such as communication security, access control, and data privacy. Grid technologies already support cross-organizational communication and security protocols since the exchange of confidential data is a usual task in the Grid domain. The basic concepts for the security architecture are published in [65] and [69].

As the prototype is build upon the UNICORE 6 Grid middleware that is designed with respect to modern security standards, we can partly profit from the Grid security capabilities provided there as basis for our security system. If another WSRF-compliant service framework is used for the realization, either an implementation of the applied standards or a similar solution must be implemented based on the below described concepts.

Grid security is based upon long-living and personalized X.509 certificates issued by a Certificate Authority (CA) or an associated Registration Authority (RA). Everyone who intends to participate in a Grid has to register with this CA. Access permissions are granted on basis of the membership in so-called virtual organizations (VO) often founded by different communities and maintained in dedicated VO management systems.

This registration process is neither applicable to OaaS scenarios with many customers nor to workflow execution in the business domain. A company, possibly having several hundred employees and high employee fluctuation, cannot send each new employee personally to a RA in order to apply for a personalized certificate. Additionally, the setup and maintenance of an own Public Key Infrastructure (PKI) is often beyond the capabilities of small and medium enterprises. Hence, a more flexible and easy to use solution with low cost is necessary. Furthermore, the Grid way of assigning rights according to the membership in VOs does not provide enough flexibility for companies. Rights shall be bound to business roles from already existing identity management systems without the need to maintain all users a second time in a Grid VO management system. These facts require a distributed identity management system which is capable of integrating local identity management systems and the possibility to grant or revoke role-based permissions in a short period of time.

The access control process requires at most two complex tasks. Firstly, we have to ensure that the user who requests access, actually is person he claims to be (authentication); and secondly, we have to check whether this user is allowed to process the targeted operation (authorization) on the targeted resource. This section presents a more detailed view on the UNICORE 6 authorization and authentication process and expands it to a more applicable solution for the Grid and Cloud workflow engine architecture.

For **authentication**, the user has to prove his identity. Therefore, he has to provide some information, such as user name and password or a certificate that is only in his possession and therewith confirms his identity. Grid middlewares usually apply X.509 certificates for user authentication. For instance, a user who provides a valid certificate issued by a trusted certificate authority has successfully authenticated with the system and is allowed to communicate with UNICORE 6. Certificates are a secure and approved way to realize authentication as it is very hard to fake valid certificates. Of course, the user must keep the private part of his certificate secret. Disadvantages are, for instance, the high costs for sending all employees to a Grid RA and that it is difficult to revoke rights because these are directly conjunct with certificate.

For **authorization**, UNICORE 6 evaluates access rules – defined as XACML [41] – that are uniquely defined for each deployed service. The default UNICORE 6 system fetches information from the xUUDB component such as the user's role and local login name (to execute compute jobs on the cluster system). The xUUDB is a very simple user database that can be shared over several sites. Since its capabilities are very limited, it is only limited applicable for cooperation scenarios. The returned user attributes are passed to the XACML access control system that renders an access decision according to the deposited rules. To allow the passing of more detailed user information such as its affiliation, department, and current role to the authorization system, we have to modify the default UNICORE 6 security architecture. More details about this are given in Section 4.2.1. Previously, we present some general properties – necessary for the security infrastructure realization – of the security standards SAML [33] and XACML [41] in the next two paragraphs.

SAML is a standard that is capable of encoding arbitrary attributes, such as roles and affiliations, into a so-called assertion. Such assertions can be issued by an identity management system. SAML also provides the capability to define fine-grained credential delegation rights, for example, to express that an entity is cleared for processing an activity on a resource during a well defined time frame. As UNICORE 6 already supports SAML, we assume that this standard is an appropriate basis to transport enriched user information from an identity management system to the Grid and Cloud orchestration engine.

The XACML standard offers sophisticated means for defining fine-grained access control rules and evaluating access decisions on these rules. A rule is described by means of the user, the targeted resource, and the desired action. Furthermore, conditions can be added that express dependencies between these elements. An XACML rule evaluator, called the *Policy Decision Point* (PDP), is capable of considering the user attributes that are part of the signed SAML assertion. The result of a rule is to either deny or grant access to the resource. By using one of several different combining algorithms, XACML allows the combination of several rules to one overall result that is the final access decision. Since

the requested and provided user attributes have to match exactly, the policy designer has to be aware of the organizational structures and roles of the participating affiliations.

The architecture allows for including access rules in the deployment descriptor (see Section 5.4) and instantiates them during the deployment process according to RQ13. Hence, each workflow designer is enabled to freely formulate new access rules for his workflows.

4.2.1 Security Infrastructure Recommendation

As recommendation for an appropriate security infrastructure that supports a distributed identity management system with low maintenance costs and the capability to integrate different identity management systems such as Active Directory or OpenLDAP we suggest Shibboleth [70]. In combination with GridShib [3] the system is able to automatically issue short-lived certificates (SLC) together with a SAML assertion that includes the user's attributes. Welch et al. describe such an architecture in more detail in [130] and [16] for the Globus Toolkit 4 and gLite Grid middlewares. The idea of issuing Short Lived Credentials with a service (Short Lived Credential Service (SLCS)) is presented by SWITCH [116].

The major advantage of this security architecture is the seamless integration of existing identity management (IdM) systems. In-house hosted and maintained IdM systems can be used to check the provided credentials and to supply user attributes. Such attributes represent, for instance, the user's company internal roles, his department, or some other user specific information. As such attributes are encoded as SAML assertion, there are no limitations for the number or the types of included information.

An SLC usually has a short lifetime of at most one million seconds (approx. 11 days), so that the user's identity including his roles becomes invalid after this period. Additionally, it is possible to reduce the lifetime, for example, to one day. This guarantees that roles and permissions are only valid for a short period and must be renewed every workday. The short period is important since a mechanism to revoke certificates is missing in our architecture. Since SLCs are X.509 certificates, the system fits in the UNICORE 6 security architecture.

Figure 4.7 shows the general overview of the security architecture. To authenticate with UNICORE 6, the user needs an SLC that includes the necessary user attributes. To obtain it, he sends a request to the Shibboleth Service Provider that in turn redirects the user to his home Identity Provider (IdP). There, he authenticates himself with his home credentials, e.g. user name and password. After this, the Shibboleth Service Provider retrieves the user attributes from the IdP, generates and signs a short-lived certificate that includes these

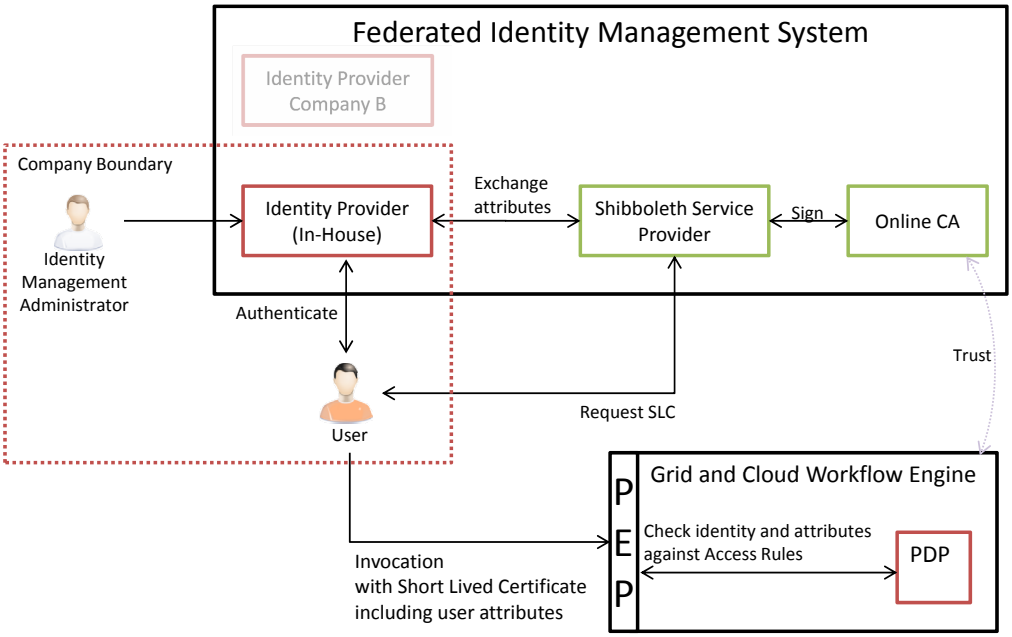


FIGURE 4.7: Federated Security Infrastructure

attributes[5]. Now, this certificate can be used for the communication with UNICORE 6. Since UNICORE 6 must be able to check the validity of the certificate, there has to be a trustful relationship between the federated IdM system (especially with the OnlineCA that signs the certificate) and the UNICORE 6 service container.

The new authorization process is similar to the original UNICORE 6 process. The user utilizes the SLC to establish an SSL connection to the prototype, denoted as BIS-Grid workflow engine, so that transport layer security as well as secure user authentication is guaranteed. During the authorization process, all requests have to pass the Policy Enforcement Point (PEP) before being delivered to the actual service. The PEP asks the Policy Decision Point (PDP) for a decision that is based on the well-defined XACML rules. In this request, the PEP includes all attributes that are attached to the SLC. Only when access is granted, the request is delivered; otherwise the user gets an "access denied" response.

Generally, it is possible to integrate many companies with the presented federated IdM system. But the integration also requires some additional effort. Each participant has to extend the locally existing IdM system to communicate with the Shibboleth service. This is an advanced task that exceeds the IT capabilities of many companies or the know-

[5] A more detailed process is given in [116]

how of a scientist. Of course, other scenarios are realizable (see below) but this one guarantees an high security level and is expandable, scalable, and flexible at the same time. Furthermore, the usage of X.509 certificates is mandatory if the tracking of indirect user activities (cf. RQ19, RQ21, RQ23, RQ25) initiated by the workflow engine is required since credential delegation is the only possibility to realize this.

The idea to integrate Shibboleth and SLCS with the UNICORE system is not completely new. In [43], Faroughi et al. present an integration with UNICORE 5.

Portal scenario

Today, portals are popular user clients. They provide browser based access to various functions as portlets – a pluggable interface to a logical subsystem. A portal is an additional component that is placed between the user and the services and so hides the user through the indirect communication. Such an additional component also requires additional effort for security, if the user's identity needs to be ensured.

If a portal is used, a valid credential delegation requires at least two delegation steps: from the user to the portal and from the portal to the used service (the BIS-Grid workflow engine in this case). All participants, the user and the portal, need valid certificates to sign the delegations. But, a portal can also help to realize a scenario without user certificates that is easier to handle for users, but with a slightly lower level of security.

The user logs in at the portal as usual, e.g. with user name and password credentials. Now, the portal tends for the communication with the BIS-Grid workflow engine. It fetches the user attributes from the local identity management system that creates a signed SAML assertion. For this and for the communication with the service, the portal uses a portal certificate that identifies the portal instead of the end-user. The authentication and authorization process is then strictly split in two parts: the portal certificate is used to authorize the portal as valid and trustful communication endpoint and the authorization decision is completely based on the user's attributes included in the SAML assertion. Of course, there must be a trustful relationship between the portal and the service.

This scenario is slightly more unsafe than the previous scenario, because the portal is a single point of failure. If it is hacked, the attacker is possibly able to obtain assertions that includes administrator privileges. With this assertion, he possibly gets access to restricted functions.

Another problem is that an assignment of service certificates is sometimes not applicable to the legal rules of Public-Key-Infrastructures. As example, the D-Grid PKI does not allow certificates for services.

UVOS

Due to the high setup costs for Shibboleth, we decided to use a simpler solution for the prototype and for the exemplary evaluation with our project partners. This solution uses SAML and user certificates for communication and user attribute encoding. In the same time, it should be possible to manage different user groups and accounts as necessary for an OaaS scenario.

The UNICORE Virtual Organisations System (UVOS), presented in Section 2.3.3, allows for the administration of user identities combined with arbitrary attributes for each identity. Additionally, hierarchical organizations can be mapped to hierarchically organized groups and attributes can be attached to groups. Group members inherit group attributes. Groups or sub-group members and attributes can be managed by different administrators. All UVOS-managed information can be queried by SAML2-compatible applications [33]. UVOS responds to such a query with a signed SAML assertion including all attributes (group belongings, group attributes, global attributes, and user attributes). The combination of UVOS and UNICORE 6 is fully integrated and well-tested during the Chemomentum [109] project, in which UVOS was developed originally.

The usage of UVOS instead of a real federated identity management system has some disadvantages, for instance:

- The missing integration of already existing companies' identity management systems. Thus, the maintenance of user roles and attributes is necessary twice.
- Generally, each user needs a valid permanent X.509 certificate to communicate with the Grid and Cloud workflow engine and to retrieve his attributes from the UVOS system. Although, it is possible to query UVOS with only user name and password credentials, a valid certificate to communicate with UNICORE 6 is still required. Thus, all users still have to register at the RA.

4.2.2 Confidentiality

After discussing authentication and authorization, we have to consider privacy, too. If several companies work with the same workflow engine, e.g. in an Orchestration as a Service scenario, it must be ensured that only authorized users can see what workflows are deployed or what workflows are currently running. Authorized, in this case, does not necessarily require that the user has the right to completely use the workflow or management service. He is possibly only allowed to retrieve monitoring information. This authorization decision must be very fine-grained and based on the user's attributes. XACML policies are not sufficient for this issue since they control access to service methods but are not applicable to information filtering according to the user's identity.

The Workflow Management Service and Workflow Service support means to filter information during search operations. These filters apply the user's authentication attributes to information stored in the instances, for example, the instance creator's distinguished name. Instances of the Workflow Management Service store the creator's distinguished name, his affiliation, and his business role. If someone ask for all deployed workflows, the Grid and Cloud orchestration engine will only show workflows matching the same affiliation and business role. Both must be included in the user's authorization and authentication security tokens. A similar filter also guarantees privacy for searching for instances of a Workflow Service. The rules for this filtering are currently hard-coded in the prototype but a configurable filtering is also realizable. The actual access to these instances to retrieve detailed information is secured with XACML access control rules.

Additionally, it is desirable that even the requests for the WSDL descriptions are secured by some kind of policy so that only authorized persons are able to view internal details.

To sum up, the architecture provides security aspects on all layers. Besides technical and organizational issues discussed in this dissertation, the realization of an appropriate security infrastructure also requires to consider legal issues that are beyond the scope of this work.

4.3 Integration of Grid and Cloud Services

The main goal of this thesis is the secure integration of various service types as Grid or Cloud services. Such services are possibly stateful or support only proprietary security mechanisms. This section includes two parts. It firstly presents a WS-BPEL pattern to handle stateful services according to the WSRF standard and secondly it describes the plug-in mechanism for the realization of successful communication with various partner service types.

4.3.1 BPEL Pattern for WSRF-compliant services

The WSRF specification [103] defines in WSRF-RL (Resource Lifetime) [114] the life cycle of a WSRF resource (instance). Such a life cycle can be partitioned in three phases: instance creation, instance usage, instance destruction. Figure 4.8 shows the WSRF life-cycle graphically.

The standard interfaces for WSRF-RL mirror this life cycle by offering operations for renewing the termination time or destroying the resource immediate or scheduled. However,

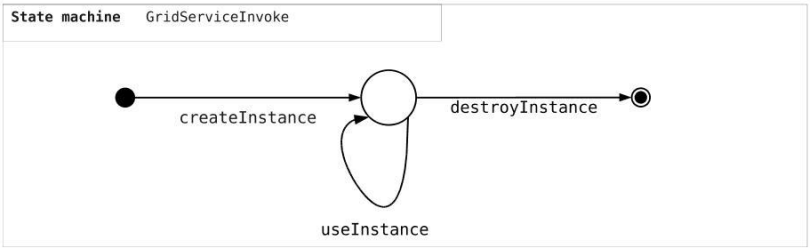


FIGURE 4.8: WSRF service utilization

a default interface for instance creation is omitted in the WSRF specification because factory requirements would be too varied [114]. Hence, instance creation operations differ in their exact interface but they all target the same function.

The general pattern to handle Grid services with pure WS-BPEL means is described in Table 4.3 and published in [24]. An extended version with all technical details can be found in the project deliverable [68]. The development of the Grid Service Invoke pattern is based on observations of the SOAP message exchange between clients and Grid services as well as on a similar solution described by Ezenwoye et al. [42].

As example, we implemented the Grid Service Invoke pattern in WS-BPEL for both the UNICORE 6 middleware and for Globus Toolkit 4 (v4.0.5). It uses a simple counter service, which is started with an initial integer value that can be arbitrarily incremented during service lifetime. Globus Toolkit 4 provides such a service and a corresponding client by default. For UNICORE 6, we implemented an analogous service and client. The general procedure is that after the creation of a Grid Service instance via a corresponding factory service, information referencing the instance is returned. This information is then used to create new endpoint references for further dynamic invocations of the Grid Service instance, for example, to invoke an operational method (increment counter) or to destroy the Grid Service instance.

The complexity of Grid service invocations in WS-BPEL is caused by the dynamic service invocation and especially by the construction of endpoint references with given WS-BPEL activities. Figure 4.9 illustrates the corresponding WS-BPEL process – within the figure, the WS-BPEL activity names are annotated in brackets. The example invokes the counter service (a value is added) twice.

The pattern illustrates that the handling of stateful services invocations for WSRF-compliant service frameworks is possible with pure WS-BPEL means. It is not necessary to extend the standard. This increases exchangeability of the WS-BPEL workflow engine and therewith sustainability of the complete architecture.

Motivation	In Grid environments, Grid middlewares are based on the Web Service Resource Framework (WSRF). Thus, services provided in Grids are WSRF services, which are stateful and require instantiation. The invocation of such a Grid service is complex and demands the execution of certain activities in order.
Intention	Define a workflow pattern that invokes a Grid service by using its factory for instantiation, using the instance, and destroying the instance.
Behavior	See Figure 4.8
Participants	The invoker of the Grid service, the Grid service factory, the Grid service instance.
Consequences	(1) Complex Grid service invocation is encapsulated in a workflow. This reduces invocation errors and redundant implementation by reuse. (2) The Grid service invocation workflow can be provided as a service itself. This simplifies the protocol of higher-level workflows and abstracts from invocation details. When the workflow language is hierarchical, the Grid service invocation may be described with the same language as the high-level workflows.

TABLE 4.3: Pattern "Grid Service Invoke"

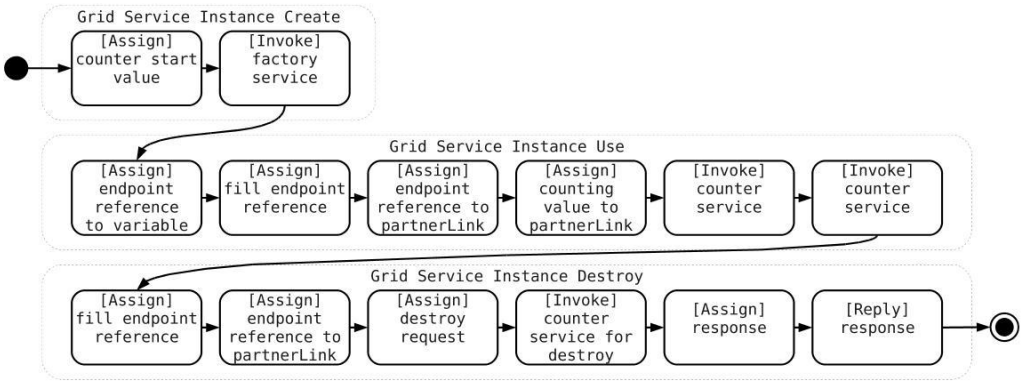


FIGURE 4.9: WS-BPEL WSRF counter service example

4.3.2 External service invocations

As already described, messages sent from the WS-BPEL engine are redirected to the BIS-Grid proxy implementation that again forwards the messages to the correct Workflow Service instance. This Workflow Service instance applies to the external service call configuration that depends on the external service type. For the WS-BPEL workflow engine, this (re-)configuration should be completely transparent. In this section, we describe the mechanism for the reconfiguration in more detail.

Cloud computing does not dictate any limitations for service providers to design their communication and security requirements. Therefore, we need a plug-in mechanism to easily extend the capabilities in communicating with external services. As general parameters for SOAP based message communication, we identified the following:

1. The content of the SOAP header that encapsulates additional information such like routing or security tokens. Therefore, it is important that the system is enabled to edit the SOAP header.

2. The communication requirements (transport layer and protocols) can be manifold. Thus, the system shall be expandable by new communication means.

The Workflow Service instance collects necessary information for both configurations from its current workflow state. This allows the adjustment of the configurations at runtime since the user can call the WS-Resource operations [52] to modify this state information.

Section 2.1.1 on page 13 describes how modern SOAP message processing frameworks work. The approach harnesses the reconfigurable handler pipelines to realize a plug-in mechanism. Each external service or service provider respectively has specific requirements for security and communication protocols. These requirements are normally well-defined and do not change over time. Hence, plug-ins that exactly target the requirements of one service provider can be developed and associated with an configuration name. The workflow designer (or at least the user) has to match the service invocations with the corresponding configuration and provide the required security tokens for instance as part of the deployment package. Perhaps these information can be collected automatically from a sophisticated service registry in a later version.

Figure 4.10 depicts the internal message processing architecture for external service invocations. The proxy passes the message to the correct Workflow Service instance. The instance looks up the correct service description in its resource properties. As index, the destination of the message is used. It can be found either in the SOAP WS-Addressing header or in target URL of the HTTP request. In our prototype, we additionally allow

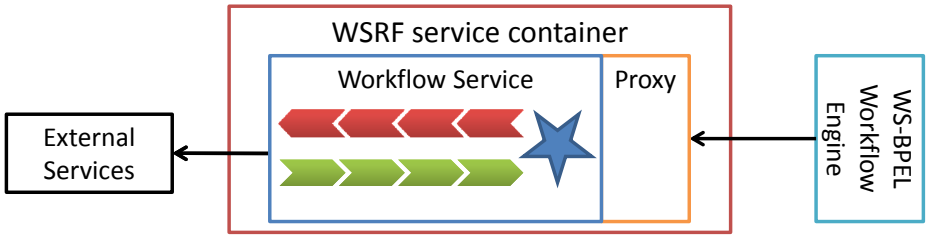


FIGURE 4.10: Handler pipeline for external service invocations

to use the service name (extracted as last section of the URL) as index for the service descriptions to be more flexible.

A service description includes three information:

1. The **service type** defines the type of the external service (for instance, insecure Web, UNICORE 6, Globus Toolkit 4, or Amazon services)
2. The **credential description ID** determines the corresponding credential configuration for this external service call.
3. An optional **endpoint reference** allows the redirection of the call to an equivalent service. This enables a late service binding and the usage of "dummy" service addresses in the WS-BPEL description.

The credential description ID is used to retrieve the credential configuration for this external invocation which are part of the Workflow Service instance state information. The credential descriptions are separated from service descriptions so that the same credential can be used to invoke different services of one or several providers. The prototype supports certificate based HTTPS authentication (UNICORE 6, BIS-Grid workflow engine, or an additional uploaded certificate), credential delegation with proxy certificates (GT4) and SAML assertions (UNICORE 6), WS-SecureConversation (implemented and tested with GT4 libraries), and simple user name and password HTTPS authentication.

Generally, the target service type defines the handler pipelines for the message exchange. But also security issues affect them, e.g. a handler that inserts credential delegation information. The architecture allows to implement simple plug-ins that reconfigures the handler pipelines according to the different requirements of the service provider. Such plug-ins are simple configuration classes that adds specialized handler into the default handler chain.

Although SOA technologies enforce platform independence, there are often some compatibility problems between the different Web service frameworks. Communication works as long as the client and the service are based on the same framework and use default

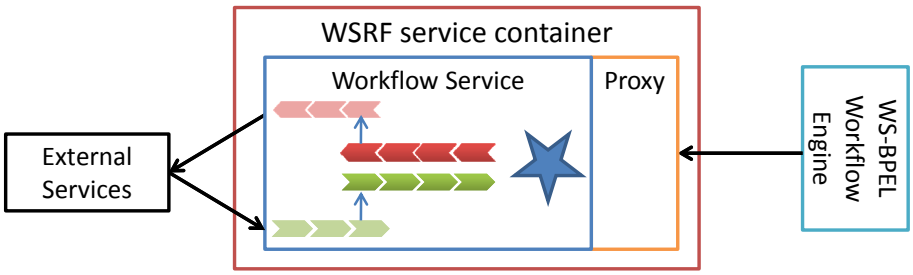


FIGURE 4.11: Integration of third-party message processing libraries

communication methods. But communication across several frameworks is more often difficult since not all frameworks are standard compliant. For instance, the Amazon Web services require that empty-tag-elements have a whitespace before the slash although this is optional according to the XML standard. Another example is the support of proxy certificates that is only available in Globus Toolkit 4 user clients.

Frameworks normally offer some kind of API or software developer kit for the implementation of services and clients that supports the respective features. A reimplementation of such features in the xFire framework that is the basis for the BIS-Grid workflow engine is too expensive. Hence, concept for reusing third party libraries is very helpful.

Figure 4.11 depicts the general idea of switching the SOAP message processing to a third-party framework. Normally, the last outgoing-pipeline handler establishes the connection to the partner. This handler is substituted by a new one that switches to the third-party message processing framework. Therefore, a mapping of the current message content and context information into the new third party message context is required. After that, a second service provider specific handler pipeline is started. The answer message need to be transformed again into the original context. This task often requires the reengineering of the third-party framework because a generic SOAP client for the respective framework is not necessarily available.

As example, we use this approach to reuse the Globus Toolkit 4 libraries for supporting the complex Grid Security Infrastructure. So we reused the original Axis handler pipeline that is already able to handle proxy certificates as well as message level transport security.

4.4 Human Interaction

The integration of people in automated workflows is an interesting topic. Although it is not central for this work, we list some possibilities to integrate people in WS-BPEL

workflows.

The creation of an asynchronous Web service that in turn serves as contact point to humans – e.g. an e-mail gateway – is an obvious way to integrate humans in business processes. But this way is very static since new activities need either a new service implementation or they are not fault tolerant since it is hard to generically validate answer messages. For scenarios with only small human interaction, such service implementations are applicable, but more complex scenarios require a more sophisticated solution that considers typical human task characteristics as delayed execution or the reassignment of tasks.

In [101], Agrawal et al. present an extension for WS-BPEL, called BPEL4People, to define a new "people activity" that is used to invoke people from the workflow in a similar way compared to a normal Web service invocation. It allows the usage of in-line – directly contained in the WS-BPEL code and all interfaces are available at the workflow engine – and standalone human tasks – given as WS-HumanTask [1] documents and executed as an external service.

WS-Human Task (WS-HT) [1] allows the design of services that represent an human activity. These services are Web services and can be invoked from a WS-BPEL engine. On the one hand, the specification allows for defining a service WSDL that represents the task interface from the invoker's point of view. On the other hand, the employees get appropriate interfaces to manage, retrieve, and process tasks. Typical management functions are the assignment of tasks to a person or a group of persons, the insertion of result information, and notification and escalation activities if a task is overdue.

Since the actual task that the human should execute is provided via a WSDL interface, the integration of human task without the BPEL4People extension directly in workflows is possible. The information that represents the task is given as WS-HT document that defines the interfaces for the respective Web service. Hence, an implementation of the WS-HT specification will suffice for the integration of human tasks in workflows. Although WS-HT uses an implicit addressing of task instances, we propose an implementation according to the WSRF standard. This will reduce the interface complexity since the task ID must not be part of each message payload.

5 Prototype

Contents

This chapter presents a short overview about the most important features of the architecture and how they are realized in the prototype, the BIS-Grid workflow engine. It focuses on the main components and the realization of the plug-in mechanism for external service invocations. Additionally, the adapter concept for supporting different WS-BPEL workflow engines is introduced. The section concludes with the description of the BIS-Grid deployment package.

5.1 UNICORE 6 service extensions

The Workflow Management Service and the Workflow Service are the main components that form the BIS-Grid workflow engine. We briefly introduce the main functions and state information as implemented in the prototype. We omit all minor important details and already presented features of the services. The project deliverables [64, 63] and the project website[1] provide additional details.

5.1.1 Workflow Management Service

The Workflow Management Service is a stateful Grid service that offers means to deploy and administrate workflows. Figure 5.1 shows the factory and the actual service interfaces as class diagram.

[1] `http://bis-grid.sourceforge.net/`

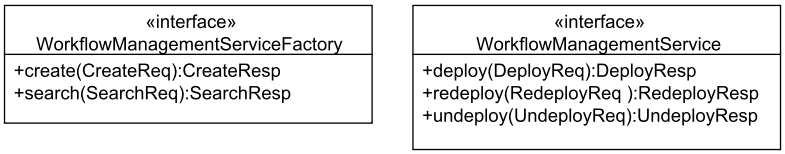


FIGURE 5.1: Interfaces - Workflow Management Service

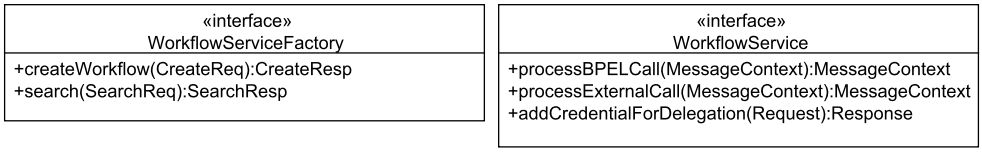


FIGURE 5.2: Interface - Workflow Service

As described in Section 4.1.3 on page 69, the factory provides an interface for creating and searching for workflows. The response of a creation call is the endpoint reference to the newly created service instance. The result of an search operation is a list of available workflows the invoking person is allowed to see. The actual Workflow Management Service comprises operations for workflow deployment, redeployment, and undeployment, as well as the default WSRF operations to manipulate the life cycle or resource properties (not shown in Figure 5.1).

During deployment, the WSDL files and the WS-BPEL file are modified to meet the WS-BPEL/WSRF instance mapping problem. Both, the original and the modified files, are provided as resource properties so that both are accessible by the workflow designer. Of course, the endpoint reference of the corresponding Workflow Service that represents the executable workflow is also available as resource property.

5.1.2 Workflow Service

The Workflow Service is the core component for workflow execution. For each deployed workflow, a new Workflow Service is created and deployed in the UNICORE 6 service container. The factory offers similar methods as the Workflow Management Service factory. It enables the user to instantiate new workflows and to retrieve a list of his workflows.

The interfaces for both, factory and service implementation, are depicted in Figure 5.2.

The class diagram omits the default WSRF operations for both services. Usually, factory services are stateless, since their dependency to the actual service is static. Since the workflow name is not fixed, we implemented a stateful factory service that manages the dependency to the actual service as resource property.

The `processBPELCall`-method is invoked if the incoming handler pipeline identified the message as workflow execution message. It gets the whole message context and processes the forwarding to the WS-BPEL workflow engine. Since there is no actual implementation for workflow execution messages, we adjusted the incoming handler pipeline for faking the existence and then invoking this generic forwarding method.

The second method, `processExternalCall`, is used to pass messages from the proxy to the Workflow Service instance. The Workflow Service creates new pipelines according to the external call configurations and executes the actual service invocation.

The third method is a default implemented service method that analyses the security tokens and stores the included delegation credentials for a later reuse in external invocations.

The Workflow Service prototype offers various information as resource properties that are interesting for the user who executes the workflow. The most important ones are:

- The WS-BPEL engine (type, host, and port) that is used for the actual execution.
- The endpoint that represents the workflow at the WS-BPEL engine.
- The current service and credential descriptions that are used for external service invocations. The user can reconfigure this properties before or during workflow execution.
- The current workflow execution state if a monitoring adapter (see Section 5.3) for the used workflow engine is available. The state includes the overall workflow state (running, faulted, completed) as well as state information for each BPEL activity in the workflow including possible error messages.

5.2 External Service Plugins

In Section 4.3.2, we have already presented the general idea of configuring the handler chain for the integration of various kinds of services. Here we present some technical details.

For a successful message exchange between the BIS-Grid workflow engine and the external service, the system has to support the partner's communication and security protocols

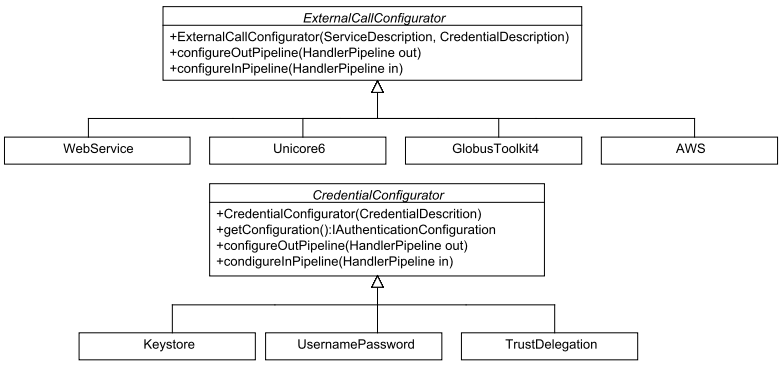


FIGURE 5.3: Configuration of external service calls

and to provide the necessary credentials to get access to the external resource. The information is retrieved from the current instance's state and the handler pipeline is adjusted according to these configurations.

Figure 5.3 depicts the abstract classes that are responsible for these configurations. A `CredentialConfigurator` is responsible for configuring and providing the credentials. The `IAuthenticationConfiguration` is used to configure the HTTP(S) client. Furthermore, the class provide means to insert additional handlers into the handler pipelines which add credential delegation tokens for instance. This configuration is only applicable if the default UNICORE 6 libraries suffice for invoking the partner service. Otherwise, the configuration information is passed to the third-party implementation which handles the security configuration.

An `ExternalCallConfigurator` modifies the handler chain according to the type of the partner service. Therefore, we need one implementation for each supported service type:

WebService

This configuration allows the invocation of simple Web services without any security requirements.

UNICORE 6

The support of UNICORE 6 services is rather simple because the Workflow Service is based on UNICORE 6 technologies and therewith all necessary functionality is already available. Since UNICORE 6 requires secured communication and at least certificate-based user authentication, the a `Keystore` is urgently required.

Globus Toolkit 4

Message exchange with the Globus Toolkit 4 middleware demands the reuse of third party-libraries since both Grid middlewares, UNICORE 6 and GT4, are based on Web services technologies but not on the same security protocols. GT4 applies mechanisms based on proxy certificates and the Grid security infrastructure. Therefore, we integrated the GT4 client technology directly into the message processing pipeline. A new handler copies the xFire message context into an Axis call object and creates a new Globus-specific handler pipeline. This pipeline is then adjusted according to the credential configuration and processes the actual call.

Partly, both technologies are based on the same libraries e.g. a library for XML signature. Since both implementations are only running with exactly one but different version of the same library, we must not only switch the message processing technology but also the Java libraries context. A new class loader re-bootstraps all necessary classes from the GT4 libraries for establishing the new context just before switching to Axis.

Amazon Web Services

The Amazon Web Services (AWS) provide a diversity of services. The portfolio range from payment and advertising services to the hosting of high performance compute resources and the allocation of human intelligence. All these services are accessible via RESTful and SOAP-based Web service interfaces. After registering with the AWS, the user can obtain two different security credentials, either a user ID and password, called *Access Key ID* and *Secret Access Key*, or a downloadable certificate. The user proofs his authenticity by signing at least the message body and a time stamp with one of the secret credentials.

We solved the support for the AWS by implementing a new handler that signs the message according to Amazon's requirements using the certificate and libraries for WS-Security [85].

Another handler previously reformats the message body to face the non-standard-compliant requirements of the AWS XML parser: Empty-element-tags must include a whitespace before the slash although it is optional in the XML specification. The handler rewrites the message so that it can be used for signature.

5.3 Adapter Concept

To support different WS-BPEL workflow engines, the usage of engine specific functions that are not part of the WS-BPEL specification is mandatory. Each engine supports dif-

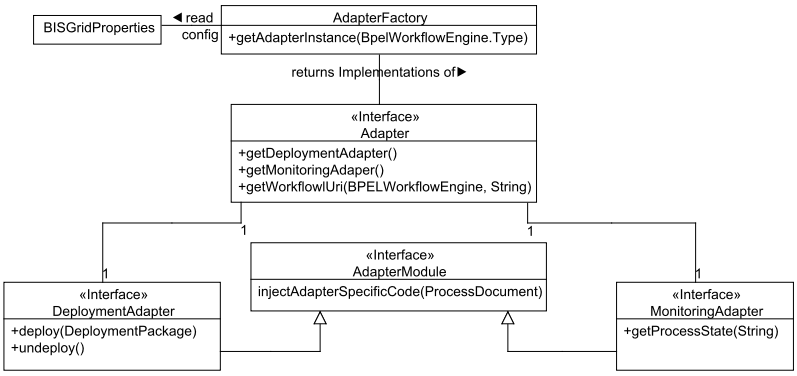


FIGURE 5.4: Adapter Concept

ferent deployment package formats and means to deploy such a package. As example, the *ActiveBPEL* engine offers a Web service interface for deployment, but no means for undeployment; the *Sun-BPEL-Engine* offers no Web service interfaces for both operations but a command line tool shipped with the application server.

Some engines offer means for monitoring that are also applicable for debugging issues. For instance, *ActiveBPEL* provides an administration Web service interface that allows for querying for a list of all (available) workflows and for more detailed monitoring information. The latter requires the internal *ActiveBPEL* workflow identifier to directly address the corresponding workflow. Section 4.1.5 describes the ID exchange between the *ActiveBPEL* engine and the Workflow Service. The pattern requires that the WS-BPEL workflow engine enables the retrieval of the process ID as part of the workflow execution, e.g. as engine specific WS-BPEL extension. Since such extensions depend on the workflow engine type, the actual pattern slightly differs for each type.

We realized all engine specific functions – usage of additional provided Web services, command line tools, and the injection of engine specific WS-BPEL or WSDL modifications – in so-called adapter modules (see Figure 5.4). Each `AdapterModule` is responsible for a special purpose such as monitoring and deployment. Generally, the deployment module is mandatory for each supported WS-BPEL workflow engine while other modules are optional. The sum of all adapter modules forms an `Adapter`. The WS-BPEL specification only state that the workflow must be published as Web service without defining details. Hence, an adapter also provides a method to retrieve the URL of the workflow.

5.4 Deployment package

The deployment package includes all information that is necessary to manage and execute one workflow. It consists of the following files:

- Deployment descriptor: The file includes all configuration parameters for the workflow. This parameters are:
  - The process name.
  - The filename of the WS-BPEL description.
  - A list of partner links. An invoked partner link (`partnerRole`) can either be static or dynamic. Static means that the partner is predefined at design time and the corresponding endpoint reference is given in the deployment descriptor. Dynamic means that the partner's endpoint will be calculated during runtime. Additionally, the client partner link (`myRole`) defines the name of the service that represents the workflow at the WS-BPEL workflow engine.
  - A list of all referenced WSDLs, XML Schemas, and other files: These entries maps the imports in the WS-BPEL document to the actual location of the files (inside the deployment package or as downloadable URL reference).
  - A list of service descriptions (see below).
  - A list of credential descriptions (see below).
  - A list of access rules (see below).
- WS-BPEL process description: The WS-BPEL code of the workflow.
- WSDL files: All WSDL files referenced in the deployment descriptor as local.
- XML Schema files: Schema files referenced in the deployment descriptor as local.
- Other files: It is possible to insert all kind of files into the deployment package to make these files available at the services (e.g. keystore or XSLT transformation files). If these files are not directly conjuncted with the WS-BPEL code, they must not necessarily referenced in the deployment descriptor.

The service descriptions and the credential descriptions form the default configuration parameters for all workflow instances with respect to the invoked services. The access rules define the access control policy for the new services in the BIS-Grid workflow engine:

A **service description** defines the service type, e.g. UNICORE 6, Globus Toolkit 4, Amazon Web Service, or default Web service. Additionally, it provides the ID of the credential descriptions necessary to communicate with the service. The Workflow Service instance

reconfigures the handler pipeline according to the service type. Optionally, it includes a new service endpoint for the redirection of the message at runtime.

A **credential description** defines the type of credentials that are necessary to communicate with the external service. The prototype allows the description of the following credentials, each in one section of the description:

- **User name** and **password** (e.g. for sending these credentials in the HTTP header)
- A **keystore** description that points to a Java keystore including a user certificate as well as possible trustful certificate authorities' certificates (e.g. for certificate-based HTTPS connections).
- An ID for a previously deposited **credential delegation** token (SAML assertion or proxy certificate). As ID, the issuer's distinguish name is used. The delegation tokens have to be previously deposited at the instance.

It is also possible to combine several sections of the credential delegation. As example, a UNICORE 6 service with credential delegation needs a certificate-based HTTPS connection and a SAML assertion as part of the message header.

The third section provides XACML [41] **access rules** that control the access to services. These rules are inserted into the UNICORE 6 authorization module during the workflow deployment process. The deployment algorithm ensures that the rules are restricted to either the Workflow Service or the Workflow Service Factory before it instantiates them. This prevents the unwanted installation of rules that affect other services.

6 Evaluation

Contents

The architecture is designed with the goal to enable the orchestration of secure Web, Grid, and Cloud services. This should ease the integration of modern service offers in enterprises' and scientific communities' application landscapes. Technologies such as Grid and Cloud computing enforces the cooperation of enterprises and institutes to realize the respective goals more efficiently. In this context, security plays a major role as in all out-sourcing and cooperation scenarios.

The design of the orchestration engine is based on the WS-BPEL workflow description language that is the de-facto standard for business workflows. The presented architecture allows the integration of all kind of SOAP-based services without creating a WS-BPEL dialect. In doing so, the BIS-Grid workflow engine has to adapt to the partner's services to support their communication and security protocols.

This chapter presents the evaluation of architecture and the BIS-Grid workflow engine as prototype. The next section outlines a review of the architecture according to the requirements listed in Chapter 3. Afterwards, it examines the applicability for business scenarios in Section 6.2 with a focus on small and medium enterprises. Section 6.3 illustrates the usability of the architecture for typical workflows used in scientific scenarios. Furthermore, this section evaluates the prototype concerning its technical characteristics in Section 6.4. This analysis compares among others the performance of a secured workflow with the BIS-Grid workflow engine to a pure ActiveBPEL workflow.

6.1 Architecture Review

This section presents a review of the architecture by listing all requirements from Section 3.3 and briefly describing how the architecture meets these. The third column rates the compliance of the architecture with respect to the requirement. "+" means that the requirement is fulfilled, "∼" that it is partly fulfilled, and "−" that the fulfillment requires some future work that is beyond the scope of this thesis.

Requirement	**Review**	
	Basics	
RQ1: Standards	The complete architecture is based upon standards. Some examples are SOAP for communication; WS-BPEL for workflow execution, SAML and XACML for authorization and authentication.	+
RQ2: SOAP communication protocol	SOAP is the default protocol for message exchange. The basic technologies used for the architecture are a SOAP-based Grid service framework and the WS-BPEL that is designed for workflow orchestration of SOAP-based services.	+
RQ3: Basic Activities Support:	WS-BPEL already comprises nearly all necessary basic activities. Instead of extending WS-BPEL, we realized additionally required activities with default means. The security and communication protocols can be adjusted for each external service call. This allows the invocation of SOAP-based services with different requirements if the respective plug-in exists. The WS-BPEL pattern for WSRF-compliant services also enables the handling of stateful services.	+
RQ4: Role-based Access Control:	The proposed security architecture intends the integration with companies' identity management systems. In doing so, it allows for seamlessly reusing roles and attributes from these systems during the authentication and authorization process. The XACML-based access control mechanism enables the definition and application of fine-grained access control rules.	+
RQ5: Secure Communication:	The communication is secured with transport layer security using certificate-based authentication. For the communication with invoked services, the architecture adapts to the respective requirements that possibly demand none or other security mechanisms.	+

Requirement	Review	
RQ6: Integration of new service providers:	The usage of handler pipelines for realizing the communication with external services and the integration of service provider's client technologies allow for a relatively simple integration of new services. The prototype demonstrates this concept with plug-ins for UNICORE 6, Globus Toolkit 4, Amazon Web services, and simple insecure Web services	+
RQ7: Hierarchical Composability	The reuse of workflows is possible in higher level workflows since the workflow presents itself as a stateful Grid service.	+
RQ8: Stateful Services Orchestration	The handling of stateful services is targeted with patterns which serve as template to handle stateful services with default WS-BPEL means. This prevents from creating a proprietary WS-BPEL dialect. A sophisticated workflow editor could partly hide the complexity of these patterns.	+
RQ9: WSRF compliance	The architecture offers all workflows as stateful services according the the WSRF standards.	+
RQ10: Control-flow and data-flow modeling	Since the workflow description language is WS-BPEL, the control-flow can be modeled well and explicitly. In the sense scientific computing, the data-flow cannot be modeled explicitly. Instead, file transfers should be initiated with the explicit invocation of file transfer Grid services that again require a well-defined cooperation of several Grid services. This partly hampers the modeling of scientific workflows. Perhaps some kind of file transfer workflows or patterns could support scientists without introducing WS-BPEL extensions.	+/∼
RQ11: Workflow isolation	The architecture provide means to filter information during search operations. Additionally, the comprehensive security system allows for defining fine-grained access rules so that the workflow engine is ready for a multi-tenant usage.	+
RQ12: Scalability	We presented some ideas about load balancing that make use of the WSRF-compliant architecture. The evaluation of load balancing is future work. In Section 6.4, we outline some performance measurements of a single BIS-Grid workflow engine instance.	∼

Requirement	Review	
	Workflow Management	
RQ13: Hot-deployment	The Workflow Management Service allows the hot deployment of workflows. During deployment, a specialized version of the generic Workflow Service – with respect to the interface and functions of the workflow – is created and registered at the WSRF service container.	+
RQ14: Functional Correctness Tests	We assume that the invoked services are functionally correct. An explicit test before deployment is only of minor importance for the actual workflow execution.	−
RQ15: Workflow Execution Statistics	The architecture allows low level workflow monitoring provided as part of the resource properties. This helps to collect data for higher level monitoring tasks. The actual implementation of such higher level monitoring tools is beyond the scope of this work.	+/∼
	Workflow Execution	
RQ16: Web Service Interfaces	The workflows are offered as services in a WSRF-compliant service container with WSDL interfaces. The interface comprises WSRF, workflow, and other operations.	+
RQ17: Variables	The WS-BPEL standard defines the usage of process variables to steer the control-flow. XPath can be used as query language for such variables. It provides basic compare and arithmetical operators. The process applies imported XML types (from WSDLs or imported XML Schema files) to initialize variables.	+
RQ18: Failure Handling	Business faults can be handled by using the WS-BPEL compensation means. The handling of technical faults is beyond the standard. However, standard means as reliable messaging can be used to increase fault tolerance during workflow enactment.	+/∼
RQ19: Workflow Monitoring	If it is possible to retrieve the current workflow execution state from the WS-BPEL workflow engine (as it is possible with the ActiveBPEL engine) the Workflow Service offers the workflow state as part of its resource properties. Therefore, an adapter for monitoring collects monitoring data directly from the engine. The ”process id retrieval” pattern is used to map the two workflow instances to each other.	+

Requirement	Review	
RQ20: Persistence	Both components, the UNICORE 6 service extensions and the ActiveBPEL workflow engine, are capable of storing their state information persistently. But this guarantees not the failure free recovery of workflows after a crash.	+/∼
RQ21: Human Interaction	WS-HT defines means to map human tasks to SOAP-based services. An implementation of this specification or simpler but less flexible Web services (specialized on exactly one task) can be used to realize human interaction in WS-BPEL-based workflows. In scenarios with only little human interaction, one specialized service can used as representation for each human task.	∼
RQ22 Quality of Service Properties:	The definition of QoS properties for a workflow is very difficult since it combines services across enterprise boundaries providing different QoS guarantees. Therefore, QoS is beyond the scope of this thesis and future work.	−
RQ23 Accounting and Billing:	Accounting and billing is not directly considered yet. However, workflow monitoring and the simple adaption of the message processing pipelines allow for the gathering of accounting information. Furthermore, credential delegation allows a detailed tracking of user activities. Billing models are beyond the scope of this work but we propose on-workflow-execution or on-external-workflow-invocation models with respect to Cloud computing services.	∼
RQ24 Adaptivity:	Todays WS-BPEL workflow engines do not support online adaptivity such as repeating workflow tasks or changing process variables. Since the architecture is build upon a standard WS-BPEL workflow engine, the support of adaptivity is not possible without extending the WS-BPEL workflow engine. The redirection of service invocations tries to allow a minimum of adaptivity.	−

Requirement	Review	
RQ25 Provenance:	The collection of provenance data such as the invoked services, used credentials, exchanged messages, and timestamps can be collected during external service invocations via appropriate handlers in the message processing pipelines. For the performance evaluation, we already implemented handlers that store parts of this information.	$\sim$

TABLE 6.1: Review of the architecture

6.2 Applicability for Business Workflow in SMEs

Small and medium enterprises (SMEs) have a special need for efficient IT support since the IT directly supports the actual business field. However, IT is also expensive and the return of investment is often hard to assess. Thus, the importance of IT support is often underestimated. This often scares IT decider to invest in new hard- and software because maintenance and licenses of sophisticated IT systems that would better support the core business are often extremely expensive.

Thus, SMEs have a stronger need for the integration of externally hosted services than global players that own the power to run large data centers. This section analyses the special challenges for cross-enterprise information system integration as necessary in outsourcing scenarios. After that, we evaluate the applicability of the BIS-Grid workflow engine in two exemplary scenarios.

6.2.1 SME Information Systems Integration

The application of a SOA is still a problem that is hard to solve for SMEs because of the lack of available resources. Hence, these enterprises do not yet profit from the benefits such as the reuse of small service units, the flexible recombination of these, and the mapping of business processes to the technical system level.

In [21], the authors list several factors that affect the decision for the acquisition of Web service based IT. These are:

- size of the business (number of employees)
- the business sector (service, manufacturing, retail ...)

- the market focus (local, regional, national ...)
- level of IT expertise (is there an IT department or some IT experts among the staff)
- annual turnover
- past experience with information technologies

Beside these mainly organizational factors also technical challenges must be faced, too. Running a SOA does not only require the initial setup but also a regular maintenance and updates [123]. If the SME has no expertise in such IT administration tasks, it has to finance further resources, such as employing new personal or purchasing consultant services. Both are costly and the return of investment must be foreseeable for the enterprise's management, in particular if IT is not the core business.

In contrast to these challenges, the introduction of service-oriented architectures offers a lot of advantages: Through the three tier architecture as described in [78, 108], companies are enabled to handle access to legacy applications, databases, and other information systems with loosely-coupled functional services instead of large and static application interfaces. Such services provide only small and simple interfaces that hide the functionality and complexity of the backend systems and better fit into the business processes as mainframe systems. This will strongly decrease future costs for the technical implementation of business process and future application integration. Additionally, SOA technologies provide a basis for the integration of external services.

Since Grid and Cloud providers are extremely specialized in their products and exploit economies of scale by running large data centers, they offer services cheaper and with a higher quality of service than the small enterprise's in-house IT departments can do. Hence, the integration of such services in the enterprises' IT will prevent SMEs from purchasing expensive off-the-shelf software. Additionally, the on-demand character and the fair billing models of Cloud computing increase attractiveness and allow for handling irregularly arising load peaks effectively.

Beside these advantages, outsourcing of business relevant IT systems is still a tough topic, especially for SMEs. The integration of external services in business processes or external hosting of business crucial data is unpopular even if the external service can provide an appropriate quality of service. The opinion of the chief information officer (in SMEs often the business owner) is determinative. This requires a security concept built upon modern standards for communication and data security as well as suitable non-technical service level agreements between the service providers and SMEs.

The usage of Cloud computing services that are designed as multi-tenant architectures arise new concerns according to security as hardware is shared with possible competitors. But such multi-tenant architecture increase hardware utilization and allow for offering services on a pay-per-usage basis. Of course, Cloud services can also be installed on

dedicated hardware if more restrictive service level agreements are required but only for additional costs.

The in this thesis presented architecture is applicable for different cooperation scenarios that facilitates the support of business processes by SOA technologies (see also [65]). Figure 6.1 gives an overview on those scenarios:

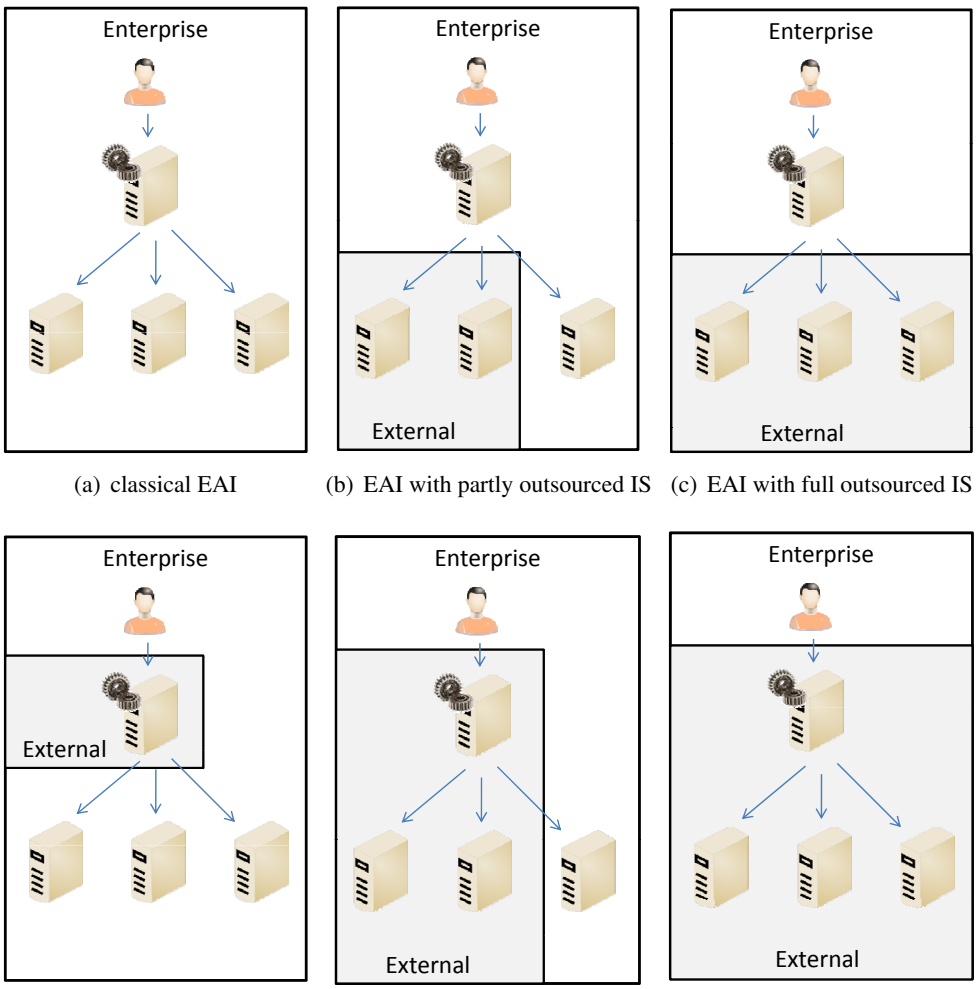
Figure 6.1(a) shows an in-house integration scenario, better known as Enterprise Application Integration (EAI) as it can often be found today. In this case, the new orchestration engine is not really necessary but it's employment permits switch to one of the other scenarios with minimal effort. Of course, it is also possible to use a default WS-BPEL workflow engine and substitute it with the secure orchestration engine later. However, architecture already provide features that are advantageously even for a pure in-house scenario such as role-based access control.

The Figures 6.1(b) and 6.1(c) depict two outsourcing scenarios where the workflow engine is hosted in-house. These scenarios guarantee that business knowledge included in the process description does not leave the enterprise's boundaries. In Figure 6.1(b), only minor important information systems are outsources and all business crucial data stays under the enterprise's administrative domain. The scenario depicted in Figure 6.1(c) presents an company with full outsources information systems only running a workflow engine in-house. This scenario seems to be improbable since the operation of a workflow engine is usually not the core competence of SMEs.

As the maintenance of a workflow engine is complex and rather a task for workflow engine specialists than for small IT departments we also consider scenarios employing an outsourced workflow orchestration engine in the sense of Orchestration as a Service:

The Figures 6.1(d), 6.1(e), and 6.1(f) depict OaaS scenarios similar to the respective above presented scenarios. The scenario in Figure 6.1(d) enables SMEs to benefit from the technical realization of business processes without introducing and maintaining a workflow engine. The next scenario (Figure 6.1(e)) is especially interesting for companies that want to outsource minor important services as well as the orchestration but keep the full control on business crucial information systems. The industrial partners of the BIS-Grid project state that this scenario matches with SMEs' requirements if the partners agree on an appropriate service level agreement.

We assume that the last scenario (Figure 6.1(f)) will be ideal for start-up companies. They profit from the elasticity and the fair billing models of Cloud computing since they can offer their services without initial costs. In the same time, it is possible to easily scale-up if the business becomes successful. Communities such like mySpace or Facebook allow promoting products to a huge number of possible customers possibly resulting in a rapidly growing number of customers. Without the availability of large data centers, such services cannot scale in an appropriate way so that customer satisfaction would decrease.



(a) classical EAI (b) EAI with partly outsourced IS (c) EAI with full outsourced IS

(d) Orchestration as a Service (OaaS) (e) OaaS with partly outsourced IS (f) OaaS with full outsourced IS

FIGURE 6.1: Usage scenarios for information system integration using a secure workflow engine

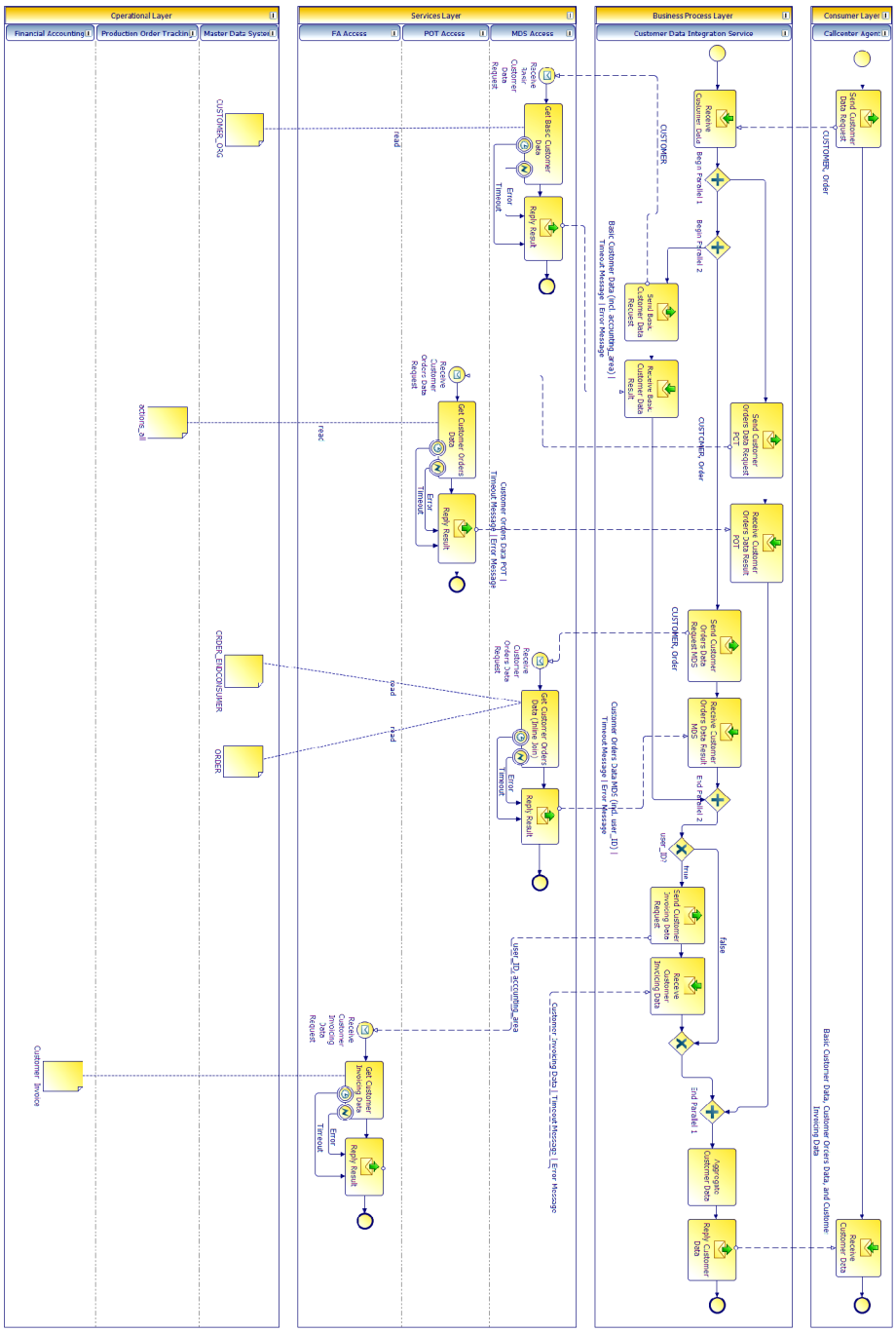


FIGURE 6.2: The call center process for read-only data retrieval as BPMN diagram. [69]

6.2.2 Example: Information System Integration

The first SME evaluation scenario examines the applicability of the architecture for an exemplary workflow with a focus on the integration of several different information systems. It originates from a real-life scenario developed by the photo finisher enterprise "CeWeColor" that was part of the BIS-Grid project. It is the number one services partner for first-class trade brands on the European photographic market supplying stores and Internet retailers with photographic products.

For the company, the impact of digital photography affected requirements to business information systems (BIS) and processes, opened up new distribution channels, and facilitated new product lines. Product mass customization and the need to flexibly respond to market development demand BIS that can adapt dynamically. The new direct distribution channel of photographic products to the customers over the Internet, the handling of customer support request became more important as the amount of call center requests grows constantly. Therefore, the company has identified the call center as one area where IT optimization and flexibility is required to lower IT costs. We applied our workflow approach [69, 55] to firstly discover the as-is and to-be state; secondly to identify an exemplary sub process that is feasible for business information system integration; and thirdly to model and to implement this exemplary process with the prototype of the architecture. Before the project, the company had neither experience in workflow modeling nor in workflow execution.

Regarding the call center scenario, information of different sources must be accessed by call center agents to provide feedback to customers – for example about order status, production failures, or accounting data. This access has to be uniform and with hard constraints to the quality of the demanded services. The exemplary process, modeled as BPMN, is depicted in Figure 6.2. The process collects information from four different enterprise information systems and provides the data as a single web service call from the user's point of view. The company implemented a graphical interface that is completely integrated with their main client application. Before that, the agent had to switch between three different applications to get access to the different information system. The continuous entering of identifier data to query the information system was cumbersome and slowed down the agent's productivity.

After the detailed modeling of the workflow as BPMN as well as the Web services and the data structures, the workflow has been modeled as WS-BPEL process within the NetBeans BPEL editor – Figure 6.3 shows a screenshot of the workflow. As input, the customer and order identifiers are required and used to get the data from the first services: production, customer, and order database. Only if the customer is an end customer and not a reseller, the data from the fourth system, the accounting database, is required.

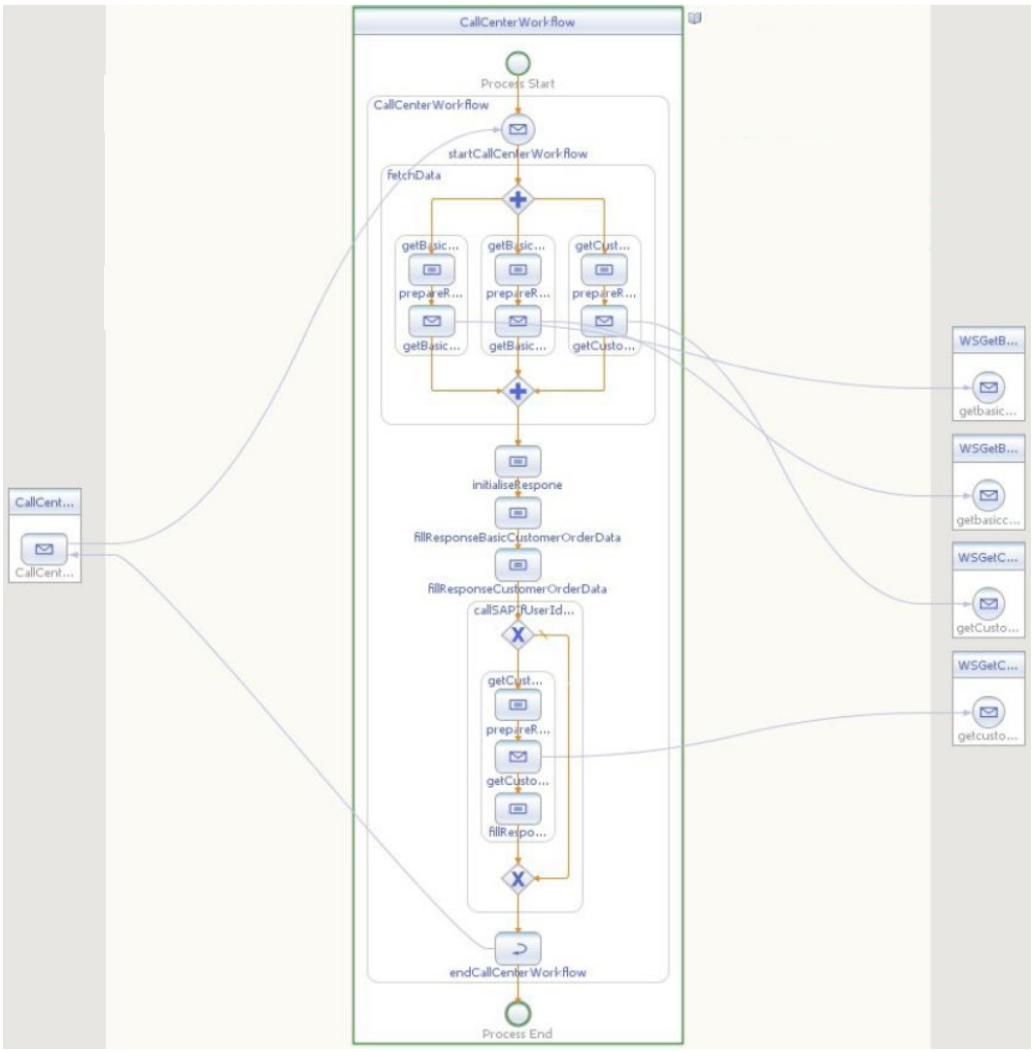


FIGURE 6.3: Example WS-BPEL workflow for information system integration [87]

fraction	$0-1$ sec	$1-5$ sec	>5 sec
$>90\%$	perfekt	acceptable	inacceptable
$>80\%$	acceptable	inacceptable	inacceptable
$<=80\%$	inacceptable	inacceptable	inacceptable

TABLE 6.2: evaluation criteria for "short response times"

During the first phase of the evaluation, the workflow was tested as in-house scenario without secured Web services and access control. During the second phase, it was evaluated according to the OaaS scenario. We tested with respect to two different locations of the BIS-Grid workflow engine installation: on a second in-house server and on the Grid cluster at the University of Paderborn. Due to serious security concerns, the OaaS scenario was only performed with dummy data in dummy backend information systems but providing the original service interfaces. These concerns originate not from the fact that the company perceives the BIS-Grid workflow engine as insecure, but the invoked services were only implemented as Web services without security.

The evaluation states that the BIS-Grid workflow engine fulfills the required functional properties with respect to the in-house and the OaaS scenario. As non-functional requirement, the company needs a "short response time" of the system so that the call center agent can quickly answer to the customer's questions. They defined time constraints for this as listed in Table 6.2 and measured the workflow response time on the locally installed prototype using the productive backend systems during normal business operations. The measurements are presented as part of the evaluation results in the project deliverable 4.3 [87] (confidential due to protect business secrets). As result, the evaluation partner stated that more than 90% of the requests were answered within one second and the remaining 10% within less than 5 seconds so that the process fits in the time constrains. Nevertheless, this evaluation must be seen critical because only one hundred workflow repetitions served as basis for these results. A detailed performance analysis of the BIS-Grid workflow engine components with different load scenarios is given in Section 6.4 of this thesis.

6.2.3 Example: Interactive Workflow

The second small enterprise evaluation partner is the world market leader in the field of wire drawing and draw-peeling for the automotive industry, named "KIESELSTEIN GmbH". Their core business is engineering and building large machines, each partly adjusted according to the customer's wishes. These individual adaptations bear challenges during the construction process.

For them, one challenge according to IT is to integrate enterprise resource planning (ERP) data and product (CAD/PDM) data that are distributed across different sites. The systems store information redundantly, since the company grew together from three different producing factories, each providing their own information systems. The different sites are connected with dedicated and therewith expensive communication lines providing only small bandwidth for security reasons.

The formal modeling of business processes as well as the technical support of such processes was completely new for the evaluation partner. Therefore, they work together with a system integrator for the machine construction industry. Both companies already cooperated before the BIS-Grid project as the integrator maintains the whole IT for the SME. Thus, a trustful relationship was already established. The system integrator supported the SME during the technical realization of the exemplary workflow process and the evaluation.

They chose the matching of articles and resources – necessary for realizing components of the total machine – between two information systems as exemplary process. Before this automation, an employee had to match the lists of such articles and resources manually. Although the introduction of the workflow partly automates this task it still requires the participation of an employee since the matching needs a lot of know-how about the respective components. The final workflow matches data from both information systems: a CAD system that manages technical drawings and an enterprise resource planning system. Both products are off-the-shelf software.

Figure 6.2.3 depicts the first half of the workflow as screenshot. Generally the workflow reads a list of all necessary parts from the CAD system and tries to find a matching counterpart in the ERP system. Not found or not matching article data is passed to the employee for manual handling. After this, the data is stored in the ERP system and again feedback is given to the employee. In the next step, the resources list – a list of articles, amounts, types etc. – is loaded from the ERP system and matched with the articles. The employee again checks this list and inserts missing data. After that, the data is stored as assembly into the ERP as well as into the CAD system. At last, the employee gets an confirmation about the successful storage operation.

The figure gives a hint of the complexity of the workflow and the high degree of user interaction. The interface on the left includes six different operations used by the user during the execution of one workflow. For simplicity of the evaluation scenario, the human interaction is realized as synchronous communications initiated by the user that returns the next required data as answer. This requires some kind of business logic also in the user client but supersedes special services for realizing human interaction as presented in Section 4.4. The interfaces on the right outline the two information systems. Not all operations are connected with the workflow because some workflow scopes are folded and therewith the connections are hidden by the editor, too.

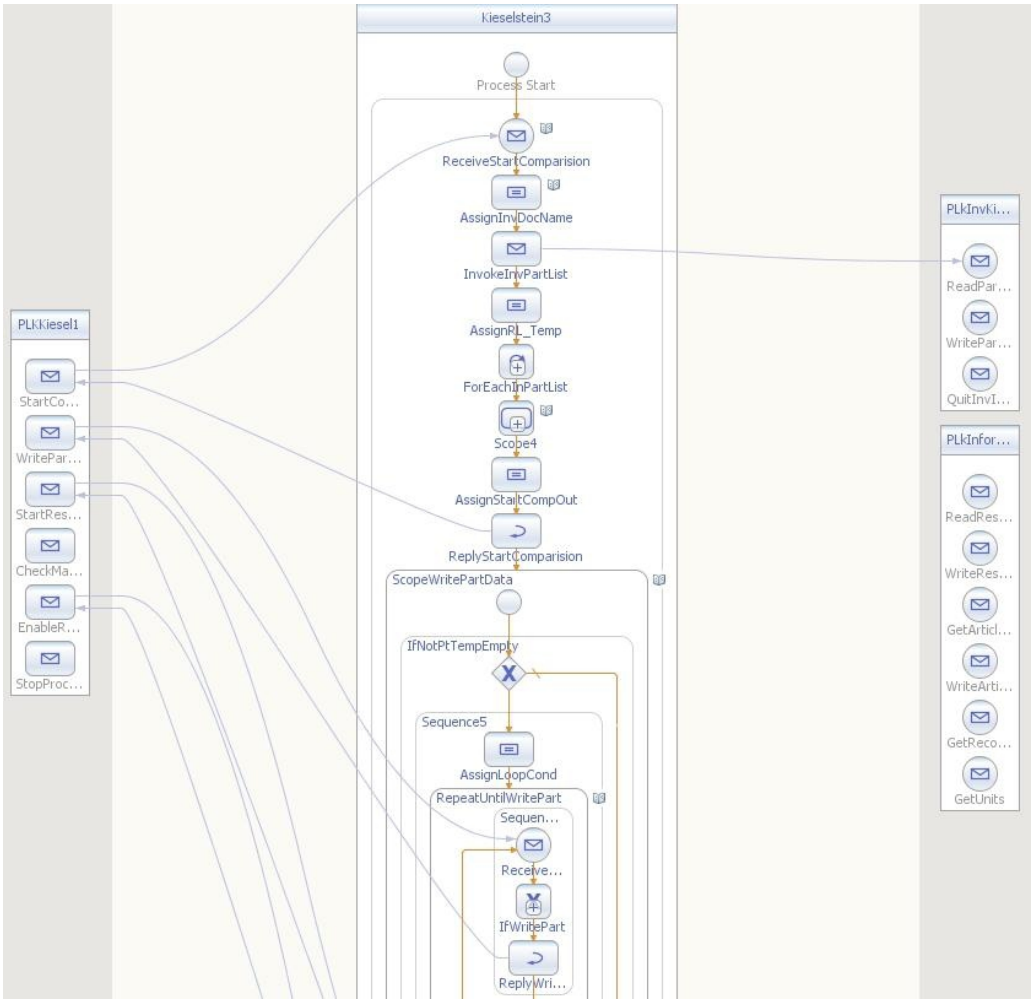


FIGURE 6.4: Interactive WS-BPEL workflow (first half) [82]

The CAD drawings as well as information about articles and resources for new machines are secret information. Hence, these data is handled very carefully. At the moment, the information systems are hosted in-house at the SME but maintained by the system integrator. Due to the trustful relationship and the complex setup and maintenance of a workflow engine, the SME would allow the external hosting of the workflow engine at the system integrator's data center. This would turn the system integrator into an OaaS provider.

The workflow has been evaluated in the following scenario. The system integrator supported the SME in workflow discovery and modeling and served as process designer. This illustrates the different roles according to the different actors in the use cases of the Workflow Management Service. Simultaneously, the system integrator provided the orchestration infrastructure in the context of OaaS. The information systems were placed in-house and encapsulated by secured Web services. Unfortunately, the ERP system offered no appropriate interfaces for the provisioning as Web service. As substitute, all ERP data was exported to a simple database. The employee of the SME who normally did the matching by hand was the user.

The evaluation result reveals that the technical workflow support saves a lot of time since the employee can process the matching within only a small portion of the original duration. But the SME also sees some drawbacks in the complexity of the required IT infrastructure (e.g. introduction of a Linux-based system in an otherwise completely Windows-based server landscape) as well as the additional maintenance. This underlines the fact that OaaS is a promising solution for SMEs. As further disadvantage, they identified the usage of personal certificates as it is usual in the Grid community. Such certificates hamper a dynamic assignment of tasks to employees. If the technical support for human interaction would be realized in a more flexible way e.g. with WS-HT [1], a dynamic task assignment would be possible even with personal certificates.

6.3 Applicability for Scientific Workflows

The applicability of WS-BPEL for scientific workflow execution has been evaluated in several papers (cf. related work about Gird Workflow Orchestration in Section 7.2). One usability problem of BPEL4WS [7], the former BPEL specification, is the parameterized execution of compute jobs (cf. Section [40] and 7.2). This problem is partly solved since WS-BPEL version 2.0 [75]. The newly introduced `<forEach>`-activity allows the execution of activities in parallel. An array that includes the parameter configurations serves as input. One parallel execution is spawned for each element of this array.

The major problem when using WS-BPEL for workflows on Grid resources is the support for several middlewares and the respective security infrastructures. To evaluate the

applicability of the Grid and Cloud orchestration architecture for scientific workflows, we implement workflows that represent typical Grid scenarios. These workflows are the submission of compute jobs and the reuse of this workflow to realize the notification after the job has finished. Beside the general applicability for the scientific domain, we also present the capabilities of orchestrating stateful services and the integration of both Grid middlewares UNICORE 6 and Globus Toolkit 4.

We only illustrate the general applicability to execute scientific tasks with WS-BPEL workflows and the reusability in more complex workflows. A comprehensive evaluation of WS-BPEL for complex experiments and modeling mainly data-flow-oriented workflows is beyond the scope of this evaluation.

6.3.1 Example: Grid Job Submission

The implementations of job submission services can be manifold. As example, UNICORE 6 support different services as the *TargetSystemService* or the *OGSA-BES* (OGSA Basic Execution Services) [46]. In this case, both services use the *Job Submission Description Language* (JSDL) [8] to define the compute task. Using job submission services results in almost generic sequences of Grid services invocations for most jobs (cf. [135]) that can be encapsulated in a generic and configurable WS-BPEL (sub-)workflow. The presented JobSubmissionWorkflow is such a workflow that represents the default submission of a compute jobs.

We identified a WS-BPEL pattern [59] for orchestrating Grid Services using standard WS-BPEL in Section 4.3.1. Based on this, we developed a workflow to encapsulate Grid service invocations for job submission to UNICORE 6. A description of this workflow is presented in Table 6.3.

Job submission to UNICORE 6 consists of several phases that are described briefly in the following. Additionally, Figure 6.5 illustrates the corresponding workflow steps. Please note that we omitted exception handling and compensation for the sake of clarity; both are not necessary for proving the general applicability of the BIS-Grid workflow engine for eScience workflows. The signature of the job submission workflow is described in Table 6.4. At least, the endpoint to a UNICORE 6 target system and the JDSL Job description is mandatory.

1. `Job Submission Receive`: A JSDL job description and configuration parameters are received and stored in process variables.
2. `Target System Service Instance Create`: A *TargetSystemService* instance is created via the default factory instance.

Motivation	Scientific workflows are often designed as non-interactive computational tasks in the form of (batch) jobs. The submission of such a job requires a well-defined invocation of several Grid services. Since this sequence can be reused in a high percentage of possible job submission scenarios it can be completely abstracted from the user. The workflow enables the submission of a compute job with only calling one service.
Intention	Define a workflow that submits a job to a UNICORE 6 installation by using the TargetSystemService and JobManagementService to submit and start the compute job given as JSDL description.
Behavior	See Figure 6.5.
Participants	The invoker of the job submission workflow, the TargetSystemFactoryService, the TargetSystemService, and the JobManagementService.
Consequences	Job submission is encapsulated in a workflow using the WS-BPEL description language. This facilitates the reuse on a higher level of abstraction and reduces submission errors.

TABLE 6.3: Job Submission Workflow description

3. `Target System Service Instance Submit Job`: The job description is submitted to the newly created *TargetSystemService* instance. One part of the result is the endpoint reference to the newly created *JobManagementService* instance.

4. `Job Management Service Instance Start Job`: The endpoint of the *JobManagementService* instance is assigned to a new partner link and used to start the actual job execution.

5. `Job Result Reply`: The workflow sends both endpoint references, the endpoint of the newly created *TargetSystemService* instance and *JobManagementService* instance, back to the invoker.

The *JobSubmissionWorkflow* submits the JSDL description to the UNICORE 6 system and does not care for the execution result or possible error. These tasks are in the authority of the user. Thus, the Grid service destroy pattern – as introduced in Section 4.3.1 – is not applied in the workflow. The endpoint references of the *TargetSystem-* and the *JobManagementService* instance are returned to the invoker for a possible reuse.

Since the anonymous submission of jobs is prohibited in some Grid installations, the workflow supports credential delegation. When the user starts the workflow, he also submits the SAML assertion that state that the Workflow Service is allowed to act on behalf

Name	*Type*	*Description*
	Input parameters	
TargetSystemFactory (mandatory)	wsa:EndpointReferenceType	The endpoint to the TargetSystemFactoryService that shall create the Target System Service instance.
JSDL (mandatory)	jsdl:JobDefinition_Type	The job description according to the JSDL standard.
LifetimeIntervall (optional)	xsd:duration	Maximum runtime for job execution. If omitted, a default lifetime is used. The maximal runtime is added to the current time stamp to define the jobs wall time
	Output parameters	
TargetSystem	wsa:EndpointReferenceType	The endpoint reference that points to the newly created target system.
JobManagement	wsa:EndpointReferenceType	The endpoint reference which points to the newly created job management that is responsible for this single job execution.

TABLE 6.4: Signature of the job submission workflow.

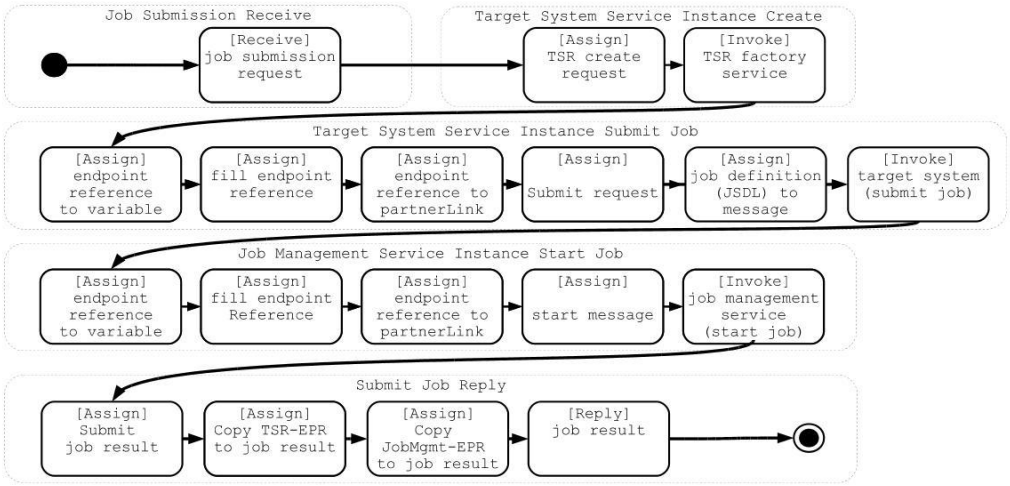


FIGURE 6.5: JobSubmissionWorkflow

of the user to the Workflow Service instance. This instance stores the assertion and uses it for all calls to the Grid site during workflow enactment.

The *JobSubmissionWorkflow* still has some potential for optimization. Each user only requires one active, reusable *TargetSystem* to submit several jobs to a UNICORE 6 site. If he has already created a *TargetSystem*, he should be enabled to optionally submit its endpoint reference instead of the *TargetSystemFactory* endpoint reference. The workflow should automatically check the service name and skip the `Target System Service Instance Create` step.

Data staging

Data staging is an important part of all scientific compute jobs. UNICORE 6 provides a *StorageManagementService* that can be used for initiating data transfer between two Grid sites. This requires the cooperation of several Grid sites and services as the *FileTransferService* is responsible for the actual data transfer and the *StorageManagementService* provide access to the file system.

A file transfer can explicitly be modeled as workflow that communicates with at least two sited. But a central orchestrator is only applicable to control instead of processing file transfers. Transferring huge data amounts through a WS-BPEL engine would cause enormous communication overhead as data must be transferred twice and will furthermore slow down the engine. Additionally, WS-BPEL is not capable of modeling or processing data streams.

As workaround, we can use JSDL capabilities to instruct the *JobManagementService* to implicitly execute file transfers as data staging. The JSDL schema includes dedicates sections to model data staging as part of the job description. In both cases, explicit or implicit file transfers, the data must be previously available at a Grid storage service.

6.3.2 Example: Hierarchical workflow composition

The workflows deployed in the BIS-Grid workflow engine are stateful Grid services. We reuse the *JobSubmissionWorkflow* in a higher level workflow for illustrating the composition abilities of deployed workflows.

In contrast to GT4, UNICORE 6 does not support notifications when jobs are done. Hence, the user has to poll regularly to get the current job execution progress. As solution, the *JobSubmissionWorkflow* is encapsulated into a second, higher level workflow that polls the *JobManagement* service instance regularly and notifies the user after the job

Motivation	UNICORE 6 supports no notifications. The client should receive a message when the workflow is finished so that it can react in a proper way.
Intention	Define a workflow that submit a job on a UNICORE 6 by reusing the *JobSubmissionWorkflow* and check the state of the workflow regularly. When the workflow is finished or faulted, send the result and the endpoint references of the participating UNICORE 6 services to the user.
Behavior	See Figure 6.6.
Participants	The invoker of the workflow, the respective services (Workflow Service Factory and Workflow Service) of the JobSubmissionWorkflow.
Consequences	The workflow offers a service that submits JSDL jobs and notifies the invoker after the execution is finished or a fault occurred.

TABLE 6.5: Job Submission and Notification Workflow description

is done. Hence, this workflow again can be reused to execute jobs in a higher level workflow that triggers new events after the previous job is finished. The workflow is described in more detail in Table 6.5.

The workflow behavior is depicted in Figure 6.6 and is composed of the following tasks:

1. `Job Submission Receive`: A JSDL job description and configuration parameters are received and stored in process variables.

2. `Job Submission Workflow Create`: The *JobSubmissionWorkflowFactory* service is used to create a new instance of the *JobSubmissionWorkflow*.

3. `Job Submission Workflow Submit Job`: The endpoint and input message to start the job submission workflow service is prepared and send to the Workflow Service instance. This triggers the actual job submission to the UNICORE 6 site as presented in Section 6.3.1. The result is the endpoint references to the *TargetSystem-* and *JobManagementService*.

4. `Check Job Execution State`: According to the WSRF standard, the current state of job execution is presented as resource property of the *JobManagementService*. After assigning the endpoint reference to a partner link the workflow repeats the retrieval of this property until the execution state changes to either FAILED or SUCCESSFULL.

5. `Submit Job Reply`: The workflow answers the invoker with the endpoint references of both, *TargetSystem-* and *JobManagementService*.

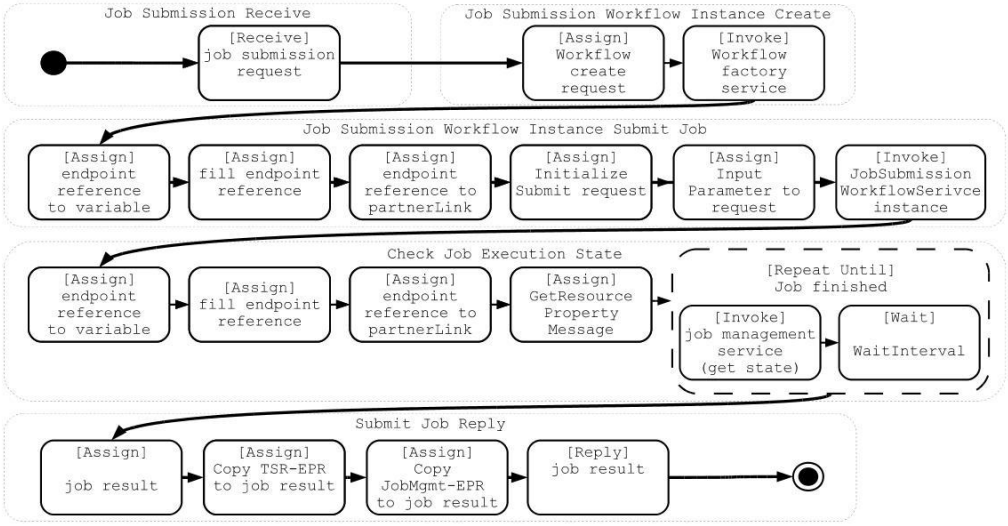


FIGURE 6.6: Job Submission and Notification Workflow

Note that we omit the destruction of the workflow instance in spite of it would be reasonable. The interface of the workflow is the same as for the pure job submission workflow (cf. Table 6.4).

Both workflows represent only a prove of concept for Grid services integration, hierarchical composition of workflows, and the feasibility to model typical scientific computing tasks with the BIS-Grid workflow engine. They still offer some improvement capabilities.

Firstly, adding optional input parameter allows for redefining the length of the wait interval between JobManagementService state retrievals. Since only the user can estimate the job execution time, it saves a lot of state requests or wait time (for short running jobs). The introduction of an initial wait interval before regularly asking for the execution state would also be helpful.

Secondly, the current workflow description uses a synchronous communication pattern – for simplicity reasons – to invoke the job submission and notification workflow. Since the actual job execution will take a while (minutes, hours, or even days), the connection will fail because of the TCP timeout. This does not affect the workflow execution but the answer message is discarded. Asynchronous communication would me more sophisticated but would require a Web service port on client side.

Thirdly, we have to consider that the job exceeds the default lifetime of a JobManagementService instance. At the moment, it is laborious to get this instance regularly and set a new

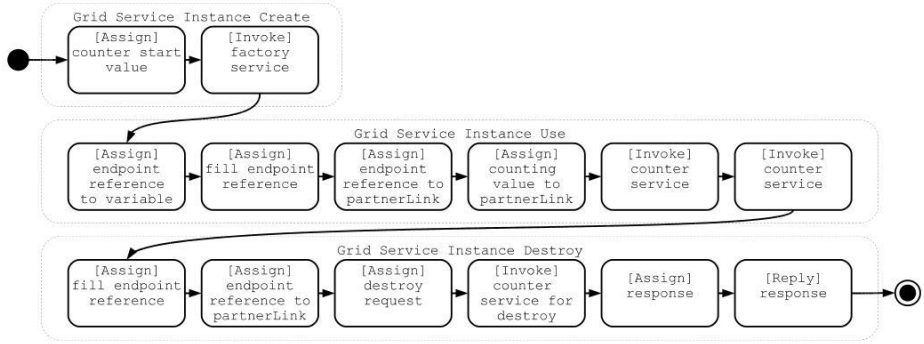


FIGURE 6.7: GT4 Counter Workflow

termination time. The workflow could automatically check the current execution state after a "wait"-period and increase the termination time, if necessary.

Fourthly, the user assigns the Grid site that is used for job execution. As a Grid normally provide some kind of monitoring about job queue length for each site, a higher level workflow could query such a monitoring service and submit the job to the Grid site with the shortest queue.

6.3.3 Example: Globus Toolkit 4 integration

With the integration of UNICORE 6, we already outlined the feasibility of the BIS-Grid workflow engine to orchestrate stateful Grid services and the usage of the WS-BPEL orchestration language for basic scientific workflows, such as the submission of compute jobs to a Grid site. As a second Grid middleware, we also evaluated the feasibility of orchestarting Globus Toolkit 4 (GT4) services. As described in Section 2.3, the two middlewares differ in several points:

- GT4 offers different **services** for the same typical Grid computing functionality.
- The encoding of the **instance identifier** in the endpoint reference is different.
- GT4 uses the so-called **Grid Security Infrastructure** (GSI) that basically differs from the UNICORE 6 security infrastructure.

Since we already outlined the usage of Grid services for job submission, we implemented only a demonstrator workflow that generally illustrates the usage of GSI secured communication and the handling of GT4 endpoint references.

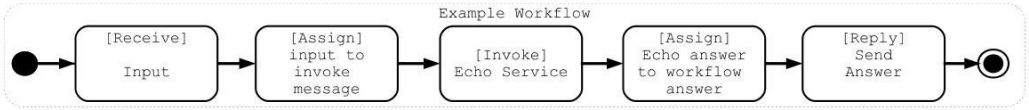


FIGURE 6.8: Example Workflow

For this, we implemented a workflow that simply invokes the GT4 *CounterService* that is shipped with the GT4 installation as depicted in Figure 6.3.3. The workflows allies the three basic steps on the *CounterService*: `Grid Service Instance Create`, `Grid Service Instance Use`, and `Grid Service Instance Destroy` corresponding to the typical WSRF usage pattern in Section 4.3.1 on page 87.

We tested the workflow without and also with message-level security based on *WS-Security* and *WS-SecureConversation* (cf. Section 2.3.2 on page 27). *WS-SecureConversation* requires the usage of credential delegation to identify the original user. Therefore, we created and send a valid proxy certificate within the SOAP header of the workflow start message to the BIS-Grid workflow engine. The workflow uses this certificate for the actual service calls on the GT4 *CounterService*.

Since all workflows has been executed successfully, the evaluation illustrates that the BIS-Grid workflow engine is capable of integrating GT4 services or in more detail services that are secured by the Grid Security Infrastructure.

6.4 Performance

The evaluation scenarios have already outlined that the importance of performance depends on the workflow and the application scenario. The OaaS usage scenario requires an acceptable performance as well as the possibility to scale. Scalability is rather an issue for load balancing than for the performance of individual workflow executions.

This section presents some performance measurements to illustrate the additional costs of the BIS-Grid workflow engine architecture. Anyway, the usage of a central workflow engine for time critical orchestrations is not recommendable in most cases. A choreography approach or a client that implicitly realize the orchestration logic would save communication overhead.

The evaluation setup is as follows: Two servers (AMD Athlon 64 X2 3800+, 4 GB RAM) and an additional computer as client are used. The BIS-Grid workflow engine and additionally a default ActiveBPEL workflow engine (version 5.0.2 on a tomcat 5.5.28) are

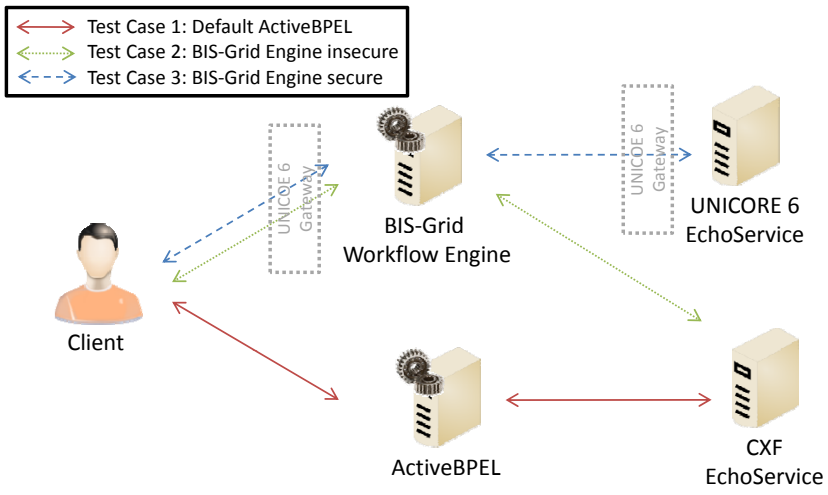


FIGURE 6.9: Test Case Scenarios for Performance Measurements

installed on the first machine. The second server hosts the invoked services on two different service frameworks: UNICORE 6 and ApacheCXF. All machines are connected with a 100 megabit local area network.

Both invoked services are *echo services* that simply return the input. These test cases allow us for comparing the performance of the default insecure orchestration to the BIS-Grid workflow engine supporting Grid security standards for authentication and authorization. The workflow the has been used in all test cases is presented in Figure 6.8. It simply invokes the external service one time and directly returns the answer to the user.

Figure 6.9 sums up the three different test scenarios:

1. The first test case is a simple, insecure Web service workflow: The workflow runs at the ActiveBPEL workflow engine and invokes a default CXF Web service (*ActiveBPEL – CXF*).

2. The second test case represents the BIS-Grid workflow engine and invokes the default CXF Web service (*BIS-Grid – CXF*).

3. The third test case uses a secured workflow: The workflow runs at the BIS-Grid workflow engine and invokes a stateless copy of the echo service secured with UNICORE 6 security (*BIS-Grid – UNICORE*).

Since the BIS-Grid workflow engine and the UNICORE 6 echo service run in a default UNICORE 6 installation, the messages have to additionally pass a UNICORE 6 Gateway. This requires an extra (in this case local) communication step. Thus, the execution of

test case 1 requires two HTTP connections; and *test case 2* requires two HTTPS and one HTTP connection; and *test case 3* requires four HTTPS connections.

We measured the duration for each workflow execution – exclusive workflow creation time – at the client. The BIS-Grid workflow engine requires the explicit creation of workflow instances. This consumes additional resources at the BIS-Grid workflow engine but the creation of workflow instances is a usual task. Nevertheless, the workflow is such simple that the ratio of workflow execution to workflow creation is not representative for real-life application scenarios. Since we disregard this fact, the presented measurements represent a worst case scenario.

For each test case, the workflow was executed 2000 times with 1, 10, 50, and 100 concurrent client threads that simulate different load scenarios. During this, we journalize the timestamps:

1. for the client message exchange
    a) just after the request arrived at the BIS-Grid workflow engine.
    b) right before the request is send to the WS-BPEL workflow engine.
    c) just after the answer arrived at the BIS-Grid workflow engine.
    d) right before the answer is send to the client.
2. for the external service message exchange
    a) just after the request arrived at the BIS-Grid workflow engine proxy.
    b) right before the request is send to the external service.
    c) just after the answer arrived at the BIS-Grid workflow engine.
    d) right before the message is send back to the WS-BPEL workflow engine
3. at the client
    a) right before the message is send.
    b) just after the answer is received

With this timestamps, we can calculate the durations for message processing at the different components of the BIS-Grid workflow engine, the WS-BPEL workflow engine, and the external service. Since we measured the timestamps with handlers only at the UNICORE 6 server, the time for the HTTP(S) communication is included in the duration of the WS-BPEL workflow engine or the external service call duration. However, since we use the same services for each workflow execution, the implementation reuses the existing TCP connections so that the overall error should relatively small.

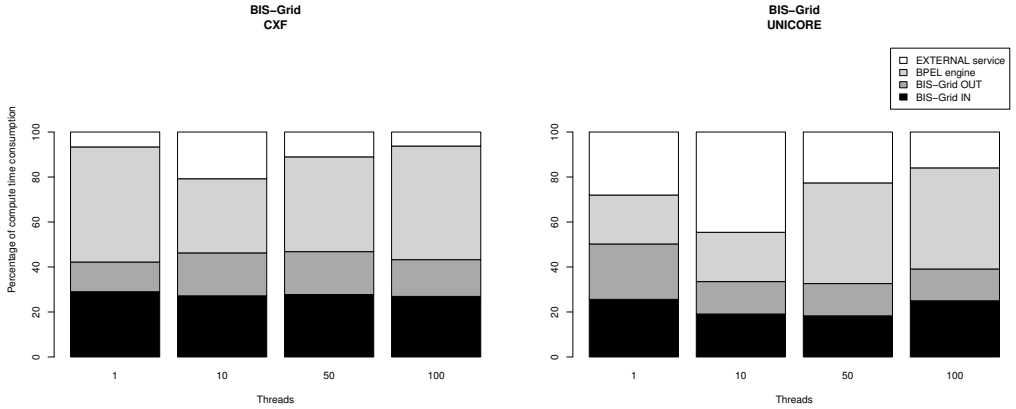


FIGURE 6.10: Medium Time Consumption expressed as a percentage

Figure 6.10 illustrates the percentage of time consumed in the different components of the BIS-Grid workflow engine and the external service for the test cases 2 (on the left) and 3 (on the right):

- *BIS-Grid IN* means the time required for forwarding a client message to the WS-BPEL workflow engine and the other way around $((1b - 1a) + (1d - 1c))$,
- *BIS-Grid OUT* stands for the consumed time for processing the in- and out-message to call the external service in the Workflow Service $((2b - 2a) + (2d - 2c))$,
- *BPEL engine* is the ratio of time consumed for all processing steps in the WS-BPEL workflow engine including the communication from/to the Workflow Service $((1c - 1b) - (2d - 2a))$,
- and *EXTERNAL service* is the fraction of time for invoking the external service including the communication from/to the Workflow Service $(2c - 2b)$.

We can deduce the following interesting facts from the resulting distributions:

- The ratios of all bars are relatively fixed (with the exception of 10 concurrent clients). This means that all services slow down in the same way if the load increases.
- BIS-Grid IN is slower than BIS-Grid OUT although both pass the message through the Workflow Service instance. This originates from the fact that BIS-GRID IN includes the time for the access control process.
- The complete processing time in the BIS-Grid workflow engine including the WS-BPEL engine (the sum of three bars at the lower end) costs only about 1 to 4 times

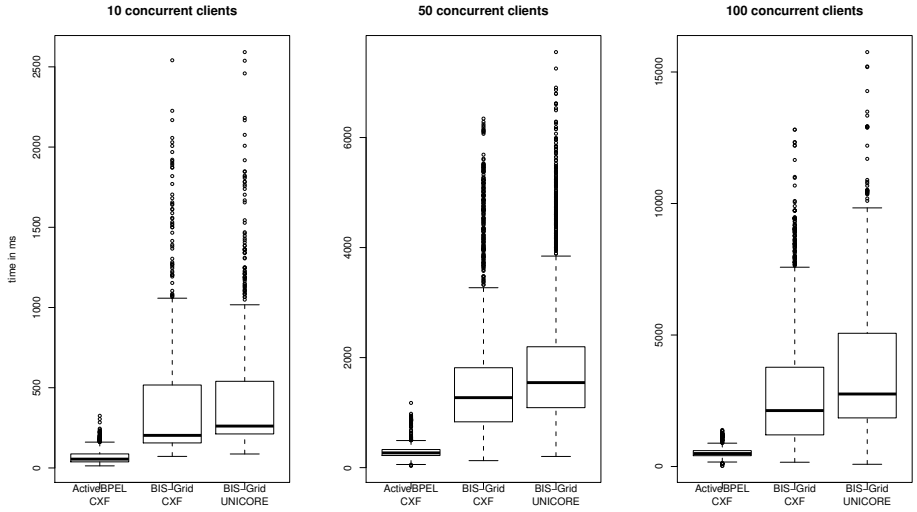


FIGURE 6.11: Response Time Distribution

more than the invocation of an simple stateless UNICORE 6 service but about 5 to 10 times more than a simple unsecured Web service.

- The ActiveBPEL engine seems to scale slightly worse than the UNICORE 6 service extension since the *BPELEngine* ratio growth disproportionate to the other components , but we assume that this originates from the handling overhead for the additional concurrent TCP connections if it has to handle more workflows concurrently.

Figure 6.11 denotes the response time distribution – measured at the client – grouped by the number of client threads as box-whisker plots [98]. Box-whisper plots depict a distribution by its five-number summary: the minimum, the lower quartile, the median, the upper quartile, and the largest observation. The whiskers (dashed lines) are 1.5 times the distance between the medium and the respective lower or upper quartile and give a hint to identify outliers.

One can observe that, of course, the test case 1 is generally faster than test case 2 and 3 but also that test case 2 is faster than test case 3. This matches with the number of HTTP(S) connections of the different test cases. Additionally, time is consumed due to the access control mechanism that is necessary once in test case 2 and twice in test case 3.

The proportion of the test cases' distributions seems similar for the depicted load scenarios, but the y-axes are scaled differently. Of course, additional load also causes higher

response times but the figure denotes that the test cases scale in a similar way. The larger diffusion for the BIS-Grid workflow engine test cases most probably originates from the worker thread architecture of Web service frameworks. Such frameworks queue incoming messages and work off the messages with a fixed number of worker threads. If the number of concurrent messages grows, also the mean wait time in such queues grows proportional. These times add up if more connections are used.

To conclude, we can state that the additional security mechanisms also causes additional costs with respect to the response times of the workflow. But these costs grow proportional with respect to the costs of an insecure workflow.

7 Related Work

Contents

This chapter presents related work grouped by three main topics. Firstly, we address some related work for simple Web service orchestration. Secondly, we outline related work in the context of Grid service orchestration and thridly we discuss some new approaches for the integration of Cloud services in workflows.

7.1 Web Service Orchestration

Within the last decade, the composition of Web services to more complex workflows has become more and more popular. Many different approaches for the combination of Web services have been tested and evaluated. Since this is a research topic for so long, we cannot list all work done in this are. Therefore, we give an overview about related work concerning common Web service orchestration with a special focus on cross-enterprise boundary composition:

In the year 2002, Ian Foster et have al. already outlined the idea to use Grid technologies for the integration of Web services across distributed, heterogeneous, and dynamic organizations (cf. [48]). As basis for this, they propose the Open Grid Service Architecture as hosting environment that is today the basis for the Globus Toolkit middleware. They have foreseen the trend towards the distribution of enterprise computing functions as a result of the need for economies of scale.

Peltz defines the term “orchestration” and compares it to “choreography” in [106]. He states that orchestration, Web services composition conducted by a central orchestrator, is rather used in enterprise internal application scenarios while choreography is feasible for enterprise cooperation. As choreography language, he presents “Web Services Choreography Interface” (WSCI) [9] because that is a collaboration extension to WSDL. It defines the message exchange between Web services that allows for cooperating without a central

instance. Furthermore, the concept of abstract BPEL processes is presented as comparable to WSCI both describe only a collaborative protocol. Concrete details on the creation of executable workflows are omitted.

Through the usage of a Web services orchestration engine that applies high security standards and flexibility to integrate various third-party services – as presented in this dissertation, enterprises are enabled to outsource IT systems. As Tilley et al. state in [123], the outsourcing of non-crucial IT services that provide no competitive advantage will enable companies to focus on what they do well, namely its core competences.

Jung, Kim, and Kung provide an overview about standard-based approaches for B2B workflow integration in [77]. The paper presents a reference model that they used to compare three approaches for B2B workflow integration:

- Workflow Systems Interoperability: The Wf-XML specification [32] is used to define a message exchange binding between two independent workflow systems, each in one participating enterprise. They note that this approach is simply to implement but it misses some important issues, like e.g. security.
- Web Service Choreography: This approach uses Web services to realize a loosely coupled integration. A workflow engine should be enabled to invoke services from diverse enterprise domains. As appropriate standards for this, they propose XPDL [50], BPML [122], WSCI [9], and BPEL [7]. This matches our concept for integrating cross-enterprise services in one business process.
- Multi-Phase Process Composition: B2B communication is realized as a coupling of private (enterprise internal) and public processes, both enacting independently. This approach requires much more implementation effort since several workflow engines must be integrated to guarantee independent management and maintenance of workflows.

In [17], Benatallah et al. describe the design for a Web service orchestration framework called *Self-Serv* based on three components: a language for process-based composition, the concept of service communities, and a peer-to-peer orchestration model. The resulting implementation uses a model-driven approach to compose services.

The authors distinguish three kinds of services: elementary services, composite services, and service communities. Elementary services are basic services that provide access to an application; composite services represent a mix of several elementary or other composite services; and service communities embody a set of services that provide the same functionality.

Service communities are realized by registering mappings between a predefined community interface and the actual service interfaces. In doing so, rather operations are mapped instead of complete WSDL interfaces to allow registrations that only comprise a subset

of the community interface. Communities are used for late service binding using multi attribute service selection policies.

Composite services are modeled with state charts but the authors outline that it is also possible to adapt other modeling languages as BPEL. Transitions can be annotated with conditions. All participating service-operations (including the new composite service) are described as the sum of all in- and out-parameters.

The enactment of services is realized by a peer-to-peer based approach. Each state in the state chart is attached with a state coordinator that is responsible for the execution of this state. After the actual execution, it notifies the coordinator of the next state that should be executed then. Coordinators only execute activities when all preconditions are fulfilled.

Mecella et al. [99, 100] present an orchestration management system for services, called the PARIDE framework. It is based on a cooperative technology layer as CORBA, EJB, or .NET. It supports the cross-enterprise orchestration of services. As modeling language for the service and orchestration behavior, they propose Petri nets.

Beside the listed papers and mostly freely available workflow engines also commercial companies provide Web service orchestration engines in their portfolios. Such engines are partly also capable of integrating services across administrative domains, but commercial vendors do only rarely publish internal architectural details. Hence, we omit commercial approaches for related work.

7.2 Grid Workflow Orchestration

Grid workflow orchestration is a common means in eScience scenarios. Some experimental setups need the execution of many compute jobs that have to be processed in a certain order. Workflows can be used to model and execute the execution of such compute job pipelines.

Akram et al. [2] identify requirements for scientific workflows – namely modularity, exception handling, mechanisms for compensation/recovery, adaptivity and flexibility, and workflow management – by the example of a protein crystallography workflow. They also describe how the BPEL language addresses these requirements, and the shortcomings of BPEL for scientific workflows. Most prominently, these are the limited adaptivity regarding workflow modifications at run-time, the lack of support for user interactions, and the need to wrap non-portable engine-specific workflow management capabilities using appropriate standards in order to use them in a portable manner. We fully agree with these statements.

Regarding the use of BPEL for Grid service orchestration, Leymann proposes BPEL4WS as foundation since it already fulfills many requirements to orchestrate services according to the WSRF standard [89]. The appropriateness of BPEL is also examined and confirmed in [28], [38], [39], [42], and [111]. These works mainly focus on scientific workflows and, except for Ezenwoye et al. [42], rely on extending or adapting BPEL, thus creating dialects. In [74], Joncheere et al. state that BPEL4WS offers some drawbacks for Grid services orchestration as the missing possibility to model cross-cutting concerns as tackled with aspect-oriented software development.

Tan and Turner present their concept to orchestrate Grid services with ActiveBPEL and Globus Toolkit 4 in [118]. They conclude that BPEL can be used to support orchestration of Grid services but they see some open issues like the incompatibility of Axis versions to deploy both frameworks on the same Tomcat, missing support of GSI in ActiveBPEL, and a complicated handling of resource keys (instance identifiers) within the workflow engine. The presented architecture of this dissertation facilitates among others the handling of resource IDs with WS-BPEL pattern.

Dörnemann et al. [38] describe how the WS-BPEL extension mechanism can be used to introduce new BPEL activities that enable Grid service communication. They introduce three new activities to support the invocation of stateful WSRF services:

- `GridCreateResourceInvoke` calls the factory of a WSRF service that creates a new resource. The resource key then stored in a BPEL variable.
- `GridInvoke` invokes the actual service instance identified by the previously stored resource key.
- `GirdDestroyResourceInvoke` destroys the service instance.

The implementation targets Globus Toolkit 4 services and supports the Grid Security Infrastructure (GSI). It is based on the ActiveBPEL 2.1 workflow engine that supports BPEL4WS [7] instead of the newer WS-BPEL [75]. As workflow designer, they extended the Eclipse BPEL editor by introducing new symbols that represent the new BPEL activities. The BPEL extensions tackle the similar problems as our WSRF BPEL pattern presented in Section 4.3.1.

The execution of jobs with WS-BPEL is also discussed in [117] in which a two-stage approach is proposed. In the first stage, a base flow is modeled to define job execution, supplemented by a JSDL job description and a fault-handling policy based on WS-Policy [127]. This base flow is expanded automatically in the second stage by additional WS-BPEL fault-handling activities corresponding to the respective fault-handling policy. The execution of the workflow requires two additional non-WS-BPEL services, a job proxy to encapsulate job execution and to receive notification messages from a scheduling

system, and a fault-handling service to apply extended fault-handling strategies such as workflow instance migration.

In [135], Zhao et al. present a visual tool that abstracts a typical sequence of BPEL activities for scientific computing to a new single activity. This sequence comprises steps like *submitTask* or *getTaskStatus* and looks slightly similar to the job submission workflow we presened in Section 6.3.1. However, Zhao et al. focus on visual complexity in the workflow editor. The proprietary code is translated to standard WS-BPEL that is then deployed in a WS-BPEL engine. This is a similar idea like hiding our *Grid Service Invoke* pattern as single activity in an sophisticated WS-BPEL editor.

The importance of the BIS-Grid workflow engine approach is strengthened since a workflow engine with similar design goals is presented in [94]. The authors describe their idea of a BPEL-based workflow engine including a load balancing mechanism. They want to solve four main issues: the support of WSRF services, dynamic and late service binding, high scalability, and friendly workflow definition. To enable the hierarchical integration of composite services, they also demand that the workflows are presented as stateful WSRF services.

However, the actual implementation is completely different compared to the BIS-Grid workflow engine. As basis, they also used ActiveBPEL but they exchanged the service invoking layer that support WSRF services. For workflow modeling, they suggest an editor that can be used to combine standard activities. These standard activities have no direct mapping to BPEL. Instead of this, an activity is defined by a proprietary code snippets before deployment that are mapped to more complex WS-BPEL code snippets. These snippets are created by domain experts and allow a simpler reuse for unskilled users. As future work, the authors plan to integrate the workflow engine with Globus Toolkit 4. The paper gives no information about security issues.

The adaption of WS-BPEL for eScience workflows in the context of the Linked Environment for Atmospheric Discovery (LEAD) is presented in [61]. The authors also state – as we do – that the diversity of Grid workflow specification languages hampers the interoperability of Grid workflow systems. They adopt the Apache ODE workflow engine and tried to let the WS-BPEL standard untouched, but party unsuccessful. The adoption requires some modifications that are implemented as plug-in mechanisms into the ODE engine and can probably easily be implemented also in other WS-BPEL compliant workflow engines. Furthermore, they tried to cover some issues with auxiliary WS-BPEL code that is injected by the XBaya workflow composer. This is a similar approach as the presented pattern injection during workflow deployment in our architecture. Furthermore, the authors emphasize that WS-BPEL is appropriate for parameter studies as it introduces the new `<for-each>` activity that allows the sequential and parallel execution WS-BPEL activities.

In [6], Amnuaykanjanasin and Nupairoj describe a solution to orchestrate Globus Toolkit services secured with the Grid Security Infrastructure (GSI). For each Grid service, a Web service proxy implementation is generated automatically when the user requests it. To overcome the GSI, Proxy Certificates are requested dynamically from a MyProxy implementation. This architecture aims on scientific workflows without considering business issues as role-based access control.

Janciak, Kloner, and Brezany present a workflow engine that is capable but also limited to orchestrate stateful Globus Toolkit 4 services supporting also the GSI in [73]. They provide the workflow engine functions as stateful Web services that manages two WS-Resources: a "Template Resource" represents a workflow description and an "Engine Resource" symbolizes a workflow instance. The functions are similar to our Workflow Management Service and Workflow Service instances. The engine uses WS-BPEL for workflow description. At runtime, it creates client stubs from the WSDL definitions to call invoked services, but no details about the internal processing and the handling of problems with incompatible service frameworks are given.

During the past years, several other Grid workflow systems – using other execution languages than WS-BPEL – appeared. Most of them are driven by a specific domain-dependent community. Here we list some famous examples:

In [83], Krishnan et al. adapt ideas of WSFL [88], one of the predecessors of WS-BPEL, to create a new Grid Services Flow Language (GSFL). They outline a framework to execute GSFL that has some common properties with the approach of this dissertation. GSDL is capable of providing the composition of Grid services again as a stateful Grid Service according to the OGSA standard [47] and the framework uses also a central orchestrator. They see large data transfers as problematic if a central orchestrator is used. As solution, they use the OGSA notification system to initiate and control peer-to-peer transfers between Grid sites.

GridAnt [5] is a Grid workflow execution tool especially designed for scientists that do not want or cannot handle complex workflow management systems. They adapt the basic ideas of the Ant build tool to define tasks (Grid Jobs) that are executed in a predefined order. The framework is capable of invoking Globus Toolkit version 2 and version 3 services. Ant tasks are normally executed locally. However, since Grid jobs are long running and the client cannot be online during the workflow execution, they propose to use a server-side service that controls workflow execution with GridAnt. The service offers an interface that allows the client to request progress information during the execution.

The KEPLER[1] workflow engine [92] is also designed especially for scientific workflow execution but is capable of invoking more than only Grid services. It provides an graphical interface to combine and configure so-called actors. Each actor processes a predefined

[1] project website at `http://kepler-project.org`

task, for instance execute a Grid job, upload a file to a Grid storage, download a website from a given URL, or doing an XSLT transformation. To flexibly support Web services, a "Web service harvester" analyzes WSDL files and creates new actors for the service. Actors are similar to plug-ins that are capable of executing one specific task. In doing so, they hide all computation logic from the user. The "actor"-concept comes from the underlying PTOLEMY II framework that allows the combination of actors with communication channels. The control-flow, in the paper discussed as "model of computation" (e.g. sequential, parallel) is generally independent from the actual actors. The additional attached "Director" component determines the control-flow and improves therewith the reusability of actors and actor groups. The framework also allows the hierarchical composition of KEPLER workflows with the help of a sub-workflow-actor.

The Taverna[2] [104] workflow system originates from the bioinformatics domain. It provides a graphical tool for creating and executing workflows consisting of processors. The *Simple conceptual unified flow language* (Scufl) is used as workflow description language. Scufl workflows allows the composition of such processors, each representing an atomic task as calling basic Web services, bioinformatic applications, or low-level Scufl workflows. Processors are connected via data links that model the data-flow between a data sources (processor output or workflow input) and data sinks (processor input or workflow output).

The UNICORE 6 middleware also provide a workflow execution engine that was developed within the Chemomentum[3] project. The engine consists of two UNICORE 6 service containers. The first represents a workflow engine that processes workflows on a logical level, the second represents a service orchestrator that transforms so-called Work Assignments into jobs, given in the Job Submission Description Language (JSDL) [8]. Both, this UNICORE 6 workflow system and the BIS-Grid engine, are implemented as service extensions to the UNICORE 6 service container. One currently existing drawback is that it is not possible to define dependencies between compute jobs, so that both jobs are executed on the same Grid site to avoid unnecessary data transfers. However, the UNICORE 6 workflow system does not support the integration of a WS-BPEL workflow engine. Some description of this workflow engine is given in [109] and on the UNICORE 6 and Chemomentum website.

The Grid workflow execution service (GWES) [67] is a workflow engine developed within the K-Wf Grid project[4]. The GEWS uses the GWorkflowDL (Generic Workflow Description Language) [4] that allows for modeling workflows as high-level Petri Nets. According to the authors, such Petri Nets are capable of modeling the control- and the data-flow.

[2] project website at `http://www.taverna.org.uk/`
[3] project website at `http://www.chemomentum.org`
[4] project website at `http://www.kwfgrid.eu/`

Both, GWES and GWorkflowDL allow the support of several different layers of abstractions, e.g. abstract operations, a mapping of these operations to available resources, and a mapping of these to a concrete workflow. As client tool, the system provides an intuitive and graphical user interface as portlet. Basically, the GWES is able to orchestrate Globus Toolkit 4 services, but the authors state that it is easy to extend. The GEWS system is used in several D-Grid[5] projects.

In [36], Deelman et al. describe the Pegasus system. As execution environment, the Globus middleware is targeted. Pegasus defines a new dedicated server – the submit host – that schedules the actual jobs to the different Grid sites. The Pegasus software stack consists of is the Pegasus services, DAGMan[120], and Condor [121]. It uses several information systems as the Globus Monitoring and Discovery Service (MDS) and the Replication Location Service (RLS) to map abstract to concrete executable workflows, both defined as directed acyclic graphs (DAG). Since Pegasus workflows can be very large, the system schedules tasks only until a dynamic scheduling horizon. This partly enables the adaption of a changing Grid environment during workflow execution. The article presents a case study that uses a real-life workflow originating from the astronomy science.

Yu and Buyya present another interesting approach for realizing Grid workflows with late scheduling in [133]. They describe their Workflow Enactment Engine (WFEE) that is capable of invoking services on Grid middlewares. The prototype uses Globus Toolkit but with some more implementation effort other middlewares should be addressable. For workflow description, Yu and Buyya introduced a new XML-based workflow description language called xWFL (XML-based WorkFlow Language) consisting of tasks. A task represent an executable defined by a name, the host, in- and output ports, data links (to model the data-flow), and parameters (key-value pairs to allow parameter studies). The "workflow coordinator" (WCO) component controls the execution at the Grid sites. Yu and Buyya propose the usage of tuple spaces for message exchange between the central WCO and the Task Managers (TM) each responsible for a single task.

7.3 Cloud Workflow Orchestration

Several upcoming and new projects that target the orchestration of Cloud services underline the relevance of this topic. The current Cloud computing hype boost the idea of outsourcing IT systems and also the reintegration of such systems with the in-house IT landscape. Hence, the idea of Orchestration as a Service becomes more popular.

[5] project website at `www.d-grid.de`

This section presents some commercial approaches for workflows in Cloud computing environments, but the companies do neither reveal internal architectural details on their websites nor publish scientific paper. Thus, we can only provide a very brief overview about those commercial products. Beside this, we also list related work for scientific approaches that integrate Grid workflow engines with Cloud services. Additionally, we list related work that deals with Cloud interoperability.

Microsoft provides the .NET Workflow Service as part of the .Net Services of the Azure Services Platform[6] in order to execute user-defined declarative workflows as lightweight service orchestrations. These services facilitate the idea of an *Internet Service Bus* that addresses the need for cross-enterprise service orchestration, supporting both the Software as a Service paradigm as well as Microsoft's Software-plus-Services strategy.

CSC[7] announced Cloud Orchestration Services and Trusted Cloud Services promising various features such as service level management, remote monitoring, reporting, data transparency, and security while ensuring industry-specific compliance and auditing services. Business Process as a Service (BPaaS) is named as one category of Trusted Cloud Services. Unfortunately, there is very little information on the concrete services, their realization, and the respective Service Level Agreements.

As a third example, Cordys also promotes cloud-based service orchestration, called Enterprise Cloud Orchestration[8]. They emphasize the still-traditional nature of the SaaS distribution model in contrast to the Cloud idea as a federation of different Clouds that may range from general-purpose Clouds to specialized Clouds in the future. Fundamentally this requires an orchestration layer in the Cloud to enable enterprises developing new business models and facilitate Application Service Provisioning.

Nowadays, the need for the integration of Cloud services originates from the business domain that already uses Clouds in a productive manner. For scientists it is still unusual to use Cloud services for eScience: The usage of commercial clouds is often not funded by scientific projects' budgets or resources for setup and managing a local Cloud are missing. Nevertheless, the scientific interest for eScience Clouds increases since the need for compute resources is enormous.

The main goal for scientists is the technical support for eScience experiments as today often realized with Grid computing. Cloud computing complements this technology. For such experiments, the availability of compute power and storage resources is most important. Thus, the usage of IaaS is much more interesting for scientists than PaaS or SaaS. In Section 7.2, we presented several workflow engines that allow the execution of scientific

[6] product website at `http://www.microsoft.com/azure/workflow.mspx`
[7] product website at `http://www.csc.com/cloud/`
[8] product website at `http://www.cordys.com/cordyscms_com/enterprise_cloud_orchestration.php`

workflows on Grid sites. Some of the developers try to use IaaS for enabling the allocation of additional resources.

In [66], Hoffa et al. uses the Pegasus Workflow engine [36] to compare different runtime environments for scientific workflow: a local machine, a local cluster, a virtual machine, and a virtual cluster. They conclude that virtual environments can tackle scalability problems in scientific workflows.

The workflow engine that is used in [107] also uses IaaS to allocate additional resources. The engine schedules tasks – modeled as DAGs according to different quality of service policies – either on a Grid, or a Cloud infrastructure, or both. This increases the probability of holding agreed deadlines.

Juhnke et al. [38] present a Grid workflow engine based on an extended ActiveBPEL implementation. In [37] the authors ouline the idea of using the Amazon EC2 Cloud for allocation of additional resources. The workflow is not used to integrate Cloud services as part of the business process, but the workflow engine supports the dynamic provisioning of additional resources if an invoked service is overloaded. The authors extend the ActiveBPEL 2.1 workflow engine with a load balancing module that measures worker nodes' load and starts new worker node instances if the load exceeds a threshold. A "dynamic resolver" that substitutes the original ActiveBPEL invoke handler is responsible for late service binding and therewith the actual load balancing. Generally, the on-demand provisioning of virtual resources is helpful to guarantee a certain degree of quality of service, but load balancing infrastructures are already included in some IaaS frameworks that provide similar functionality without the conjunction with a WS-BPEL engine. Furthermore, the load balancing only works with services that are based on virtual machine images provided by the same administrative domain.

In [76], Juhnke et al. again extend the architecture by introducing a fault tolerance mechanism that again uses the on-demand provisioning mechanisms for load balancing and additionally for masquerading technical faults. Through this, a broken service will not necessarily cause the crash of the workflow if e.g. another service instance can be used instead. This techniques require the independence of service invocations and is therefore limited to stateless services.

Wlodarczyk, Rong, and Thorsen introduce in [131] a collaboration idea that should enable enterprises to share their Cloud resources. They define an enterprise Cloud as a special form of a public Cloud with additional functions such as workflow administration, workload management, and monitoring. Such enterprise clouds are again the basis for inter-enterprise integration of information systems, which they call "Industrial Cloud". Industrial Clouds facilitate integration tasks like policies, reliability management, security and trust, outsourcing, and subcontracting.

Some other work [13, 26] stress out the need for workflow engines for collaborative services based on Clouds. However, they give no details about the concrete implementation or how to bridge different security technologies. In [12], Hewlett Packards Labs present their idea of a multi enterprise content management collaboration platform. The platform offers a SaaS cloud that allows the storage and distribution of content such as documents, files, information. They also demand the automatic combination of information to new documents with the help of workflows.

8 Future Work and Conclusion

Contents

This chapter gives an outlook onto future work and interesting related research topics. Furthermore, it concludes this thesis with a summary of the main contributions.

8.1 Outlook and Future Work

The presented architecture realizes interoperability between different Web services implementations especially considering the aspects of stateful services as well as security and communication protocols. The prototypical implementation misses some features to become a productively applicable software. The following items list a number of necessary and nice-to-have features:

- **Recovery**: If one of the two BIS-Grid workflow engine components fails, the other component should automatically stop to prevent failures because of internal errors. If the ActiveBPEL engine tries to send messages as long as UNICORE 6 is still down, workflows will fail due to connection errors. The introduction of a reliable messaging component will help to cope with such faults. An active state synchronization after the recovery is not necessary since both instances store their states separately. The Workflow Service instance fetches the workflow execution state directly from backend workflow engine if necessary.
- **Fault Tolerance**: Since the workflow engine integrates services from different vendors and service providers, a crash or unattainability of some services is occasionally possible. This demands the introduction of fault tolerance mechanisms for coping with the failure of external services. The usage of reliable messaging protocols for sending and receiving messages as well as retrying service executions or selecting equivalent services before throwing a fault will increase fault tolerance.

- **Load Balancing**: In Section 4.1.6, the thesis presents the general idea for the implementation of load balancing. For the realization, a BIS-Grid workflow engine dispatcher service (forwarding broker) is necessary that allows the cooperation of several BIS-Grid workflow engine instances. As the dispatcher must consider the stateful properties of Workflow Management Service and Workflow Service instances, it is not possible to reuse default existing Web service load balancers.

- **Automatic Service Selection**: The implementation of an automatic service selection and dispatching for invoked services would be helpful in scenarios where multiple services from different providers offer same functions. The user could define his preferences e.g. as utility functions to configure the importance of (non-)functional properties as presented in [134].

  However, the usage of Cloud services that do not reveal the actual location of the data center, raises additional legal issues. Since the data centers underlie the law of the respective country, it is a major difference, for instance, whether a European or US data center stores secret data. Thus, the system must consider Cloud-specific non-functional restrictions during the selection process.

- **RESTful services**: The importance of RESTful services is increasing especially for simple services and mash-ups. In [96], Mandel presents a way to describe RESTful services with WSDL interfaces. If it is possible to automatically generate such WSDL interfaces and map the SOAP operations into a generic REST client integrated in the prototype, the seamless integration of REST services becomes possible within workflows.

- **Human Interaction**: As the thesis has already presented in the evaluation scenario, human interaction is an urgently required use case in many workflows. Thus, the implementation of a Human Task [1] service that is integrated with the companies' identity management systems (similar to the in this thesis proposed access control system) will provide additional value for companies in the context of OaaS.

  Amazon Mechanical Turk[1] already realizes a proprietary Human Task service. It provides a Web service interface to create simple human tasks, called *Human Intelligence Tasks* or *HITs*. Everyone can execute HITs and earn the offered fee for that. However, the service is not applicable for critical business processes or company internal usage since some functions like escalation, the automatic assignments of tasks to departments or experts, and a separated multi-tenant usage for companies are missing.

[1] `https://www.mturk.com/mturk/welcome`

The listed tasks are mainly focused on implementation, but the usage of Cloud computing technologies in workflows and in business outsourcing scenarios as well as the OaaS idea offer new approaches for future research.

8.1.1 Quality of Service for Workflow Execution

Each service that is integrated in a productive business process needs to provide a well defined quality of service (QoS). This also holds for services that represent workflows, e.g. according to the OaaS model. The question on how to determine QoS properties for a workflow is interesting but also very complex if the workflow exhausts the complete power of WS-BPEL. For example, user input or service responses possibly affect the workflow execution path and impede the exact calculation of QoS properties. Furthermore, the invoked services are possibly distributed over several providers all offering different QoS properties.

Some scientists have already worked on this problem and published rules on how to aggregate QoS properties for e.g. workflow sequences, parallel scopes, and conditions [20, 72]. However, the full complexity of WS-BPEL is not covered yet. For instance, the effect of compensation or event handlers in workflows as well as user input or service responses that induce changes in the workflow execution path is not sufficiently considered, up to now. In this context, a sophisticated monitoring system that allows for exactly determining who is responsible for a fault is urgently required. The realization of such monitoring capabilities is extremely costly because of the numerous participating parties in one workflow. Thus, the relationships between the partners must be partly based on trust instead of technical control mechanisms.

8.1.2 Workflow Optimization using Cloud Interoperability

The execution of workflows on Cloud services entails new questions for service selection and workflow optimization. Cloud services are often not limited to a single country or language. For example, the provisioning of infrastructure or standard enterprise applications such as customer relationship management (CRM) are worldwide applicable. But client experience is different when using Cloud infrastructures depending on the location of the data center and the customer. Additionally, some tasks have strong dependencies such as data transfers and the corresponding job executions on this data. The relevance of this topic is underlined by the fact that some Cloud providers (e.g. Amazon) already provide the same services on different continents.

Thus, the selection and scheduling of invoked Cloud services in workflows is an interesting and complex research topic. For example, it is possibly valuable to migrate services

from a US Cloud to a European Cloud if the user profile with respect to the users' locations change over time. Another goal is to optimize the invoked services according to non-functional preferences like response time or costs. But for this, we firstly require more Cloud interoperability and the provisioning of similar services by different providers. The usage of IaaS Clouds can partly cover this if virtual machines with the same services can be deployed on differently located Clouds. Such machines can also serve as basis for PaaS and SaaS services.

8.1.3 Elastic business information systems in a Cloud

Another interesting research topic is the question whether it is possible to exhaust the full potential of Cloud computing to realize fully elastic business information systems landscapes. Cloud technologies allow for a rapid scaling of services in both directions. The demand for business information systems strongly depends on the daytime in most application scenarios. Several systems are only required during the core work time or even more rarely and can be reduced otherwise to save resources and money.

This topic is only partly related to workflows but since workflows represent the business processes, a detailed analysis of the currently running workflows can help to determine the required resources and services in the future to scale them as far as possible in advance.

8.2 Conclusion

This thesis presents a novel architecture that allows for combining Web, Grid, and Cloud services in a single workflow. The architecture is based upon process modeling and execution technologies originating from the business domain. It extends a standard fully WS-BPEL-compliant workflow engine by services that provide additional functions during the actual workflow execution. In doing so, the architecture neither extends the WS-BPEL standard nor the WS-BPEL engine itself. Instead of this, the support of manifold communication protocols and security technologies is realized with a configurable mechanism for various service types. Additionally, the mechanism allows for reusing third party message processing clients for proprietary communication protocols.

The new technology makes the integration of professionally hosted hard- and software from Grid and Cloud computing providers possible. At that, it also affects formally only internally executed business processes since it securely links a company with manifold service vendors. Therewith, it allows enterprises, especially small and medium enterprises, increasing the flexibility of business processes. Summarizing, it helps enterprises

to better support their core competencies with internal and now also external IT services.

Stateful services require a complex workflow modeling since instance identifiers must be considered for service addressing. The thesis introduces WS-BPEL patterns for handling such endpoint references without any WS-BPEL extensions. This increases platform independence since no proprietary Grid invocation dialect is introduced. We think that the complexity of the patterns can generally be hidden as one or a set of special stateful invoke activities in a graphical WS-BPEL editor. However, a too static mapping of the create, use, and destroy service invocations to one activity would hamper the flexibility of stateful service handling, e.g. for multiple invocations of the same stateful instance. As we do not rely on a new WS-BPEL dialect, the architecture is ready for workflow language advancements as it is possible to exchange the only loosely-coupled WS-BPEL workflow engine.

The presented architecture can also be applied as a basis for a Cloud service that provides workflow management and execution capabilities to its customers. The thesis describes the Orchestration as a Service paradigm which allows customers to orchestrate Web, Grid, and Cloud services by using (not running) an orchestration engine as a Cloud service. The costs for licenses of commercial workflow engines or the actual operating of such an engine often exceeds SMEs' IT departments' budgets and capabilities. An OaaS service constitutes a possibility for SMEs to benefit from the advantages of service-oriented architectures and information system integration with little initial set-up and mostly only variable operating expenses. As the OaaS service is designed as a multi-tenant Cloud service, workflows are charged on a pay-per-use basis so that it is profitable to realize even infrequent workflows. Hence, OaaS will boost new innovations since it is especially applicable for startups that directly benefit from Cloud elasticity and the combination of Cloud services without operating own IT.

Furthermore, the dissertation completes the architecture with a proposal for a security infrastructure that enables the seamless integration of companies' identity management systems with the Grid and Cloud workflow engine. The application of a federated identity management system increases the reusability of the users' authentication information without doubling maintenance costs for a second external identity management system. This is especially important if access control policies should be based on the companies' identity and role models. The application of a federated identity management system also matches to the idea of Orchestration as a Service since it eases the integration of several customers' identity management systems with one instance of the Grid and Cloud workflow engine.

The thesis evaluates the applicability of the architecture in two exemplary business scenarios. The first scenario requires an integration of three information systems realized with four services. The second exemplary workflow realizes a partly automatic matching

of articles and resources between a CAD and an ERP system for a machine constructing company. Both scenarios have different functional and non-functional requirements such as a short response time or human interaction. As result, the SMEs participating in the evaluation state that the architecture is applicable to both scenarios. Furthermore, both enterprises see the idea of using external orchestration services as reasonable, if a trustful relationship between the partners exists. Thus, the thesis illustrates the practicability of the architecture for the exemplary business scenarios. Since the it is built upon business workflow execution technologies, we can generalize that it is applicable to at least all typical WS-BPEL business workflows.

Additionally, the thesis examines the relevance of architecture for scientific workflows. The general functional applicability as realizing a job submission workflow that sketches a common task in eScience is revealed within an example workflow. Furthermore, the thesis presents an example for the hierarchical composition of workflows that allows for modeling more complex scientific experiments by reusing basic eScience workflows as parameterized standard tasks. The architecture supports different Grid middlewares and increases therewith the interoperability of these middleware implementations.

Since scientists are mostly workflow beginners but have to supervise all life cycle phases, they need appropriate design tools which hide most of the complexity. Hence, editors that allow for composing patterns or workflows with basic eScience tasks will increase the acceptance of WS-BPEL-based orchestration in scientific communities. Simultaneously, scientists will benefit from the already existing tooling around commercial workflow execution systems. The creation of such an editor is beyond the scope of the thesis.

Lastly, the thesis analyzes the prototype with respect to its performance during workflow execution with a special focus on the different components. This analysis outlines that the additional services consume extra time, but they scale proportionally compared to a pure ActiveBPEL engine.

Bibliography

[1] A. Agrawal, M. Amend, M. Das, M. Ford, C. Keller, M. Kloppmann, F. Leymann, R. Müller, G. Pfau, K. Plösser, R. Rangaswamy, A. Rickayzen, M. Rowley, P. Schmidt, I. Trickovic, A. Yiu, and M. Zeller. Web Services Human Task (WS-HumanTask). `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-bpel4people/WS-HumanTask_v1.pdf`, June 2007. Version 1.0.

[2] Asif Akram, David Meredith, and Rob Allan. Evaluation of BPEL to Scientific Workflows. In *CCGRID '06: Proceedings of the Sixth IEEE International Symposium on Cluster Computing and the Grid*, pages 269–274, Washington, DC, USA, 2006. IEEE Computer Society.

[3] The Globus Alliance. GridShib. Online `http://gridshib.globus.org/` (last visited 23.04.2010).

[4] Martin Alt, Andreas Hoheisel, Hans werner Pohl, and Sergei Gorlatch. A Grid Workflow Language Using High-Level Petri Nets. In *Wasniewski (Eds.), PPAM, in: Lecture Notes in Computer Science*, pages 715–722. Springer, 2005.

[5] Kaizar Amin, Gregor von Laszewski, Mihael Hategan, Nestor J. Zaluzec, Shawn Hampton, and Albert Rossi. GridAnt: A Client-Controllable Grid Work.ow System. In *HICSS '04: Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 7*, page 70210.3, Washington, DC, USA, 2004. IEEE Computer Society.

[6] Pichet Amnuaykanjanasin and Natawut Nupairoj. The BPEL Orchestrating Framework for Secured Grid Services. *Information Technology: Coding and Computing, International Conference on*, 1:348–353, 2005.

[7] Tony Andrews, Francisco Curbera, Hitesh Dholakia, Yaron Goland, Johannes Klein, Frank Leymann, Kevin Liu, Dieter Roller, Doug Smith, Satish Thatte, Ivana Trickovic, and Sanjiva Weerawarana. *BPEL4WS, Business Process Execution Language for Web Services Version 1.1.* IBM, 2003.

[8] Ali Anjomshoaa, Fred Brisard, Michel Drescher, Donal Fellows, An Ly, Stephen McGough, Darren Pulsipher, and Andreas Savva. Job Submission Description Language (JSDL). http://www.gridforum.org/documents/GFD.56.pdf, Nov 2005.

[9] Assaf Arkin, Sid Askary, Scott Fordin, Wolfgang Jekeli, Kohsuke Kawaguchi, David Orchard, Stefano Pogliani, Karsten Riemer, Susan Struble, Pal Takacsi-Nagy, Ivana Trickovic, and Sinisa Zimek. Web Service Choreography Interface (WSCI) 1.0. online `http://www.w3.org/TR/wsci/`, August 2002. W3C Standard.

[10] Michael Armbrust, Armando Fox, Rean Griffith, Anthony D. Joseph, Randy H. Katz, Andrew Konwinski, Gunho Lee, David A. Patterson, Ariel Rabkin, Ion Stoica, and Matei Zaharia. Above the Clouds: A Berkeley View of Cloud Computing. Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, Feb 2009.

[11] Benoit A. Aubert, Michel Patry, and Suzanne Rivard. A framework for information technology outsourcing risk management. *SIGMIS Database*, 36(4):9–28, 2005.

[12] David Banks, John S. Erickson, and Michael Rhodes. Multi-tenancy in Cloud-based Collaboration Services. Technical Report 2009-17, Hwelett Peckard Laps, 2009.

[13] David Banks, John S. Erickson, and Michael Rhodes. Toward Cloud-based Collaboration Services. In *Proceedings of Hot Topics in Cloud Computing (HotCloud 09)*, June 2009. (Online Proceedings).

[14] Tim Banks. Web Services Resource Framework (WSRF) Primer v1.2, 2006.

[15] Roger Barga and Dennis Gannon. Scientific versus Business Workflows. In *Workflows for e-Science*, pages 9–16. Springer London, 2007. ISBN 978-1-84628-519-6 (Print) 978-1-84628-757-2 (Online).

[16] Tom Barton, Jim Basney, Tim Freeman, Tom Scavo, Frank Siebenlist, Von Welch, Rachana Ananthakrishnan, Bill Baker, Monte Goode, and Kate Keahey. Identity federation and attribute-based authorization through the Globus toolkit. In *Shibboleth, GridShib, and MyProxy. In Proceedings of the 5th Annual PKI R&D Workshop*, 2005.

[17] Boualem Benatallah, Marlon Dumas, and Quan Z. Sheng. Facilitating the Rapid Development and Scalable Orchestration of Composite Web Services. In *Distributed and Parallel Databases*, 2005.

[18] K. Benedyczak, M. Lewandowski, and P. Bala. Towards a common authorization infrastructure for the Grid. In *Euro-Par 2009 Workshops: Parallel Processing*, 2010.

[19] Krzysztof Benedyczak. UNICORE Virtual Organisations Service - Overiew. `http://zam904.zam.kfa-juelich.de/svn/repo/chemomentum/uvos/uvos-server/tags/rel-1.3.1/src/main/docs/UVOSOverview.pdf`, Februrary 2008.

[20] Rainer Berbner. *Dienstgüteunterstützung für Service-orientierte Workflows*. PhD thesis, Technischen Universität Darmstadt, April 2007.

[21] R. Bose and V. Suumaran. Challenges for Deploying Web Services-Based E-Business Systems in SMEs. *International Journal of E-Business Research*, 2(1):1–18, 2006.

[22] Luc Bougé, Martti Forsell, Jesper Larsson Träff, Achim Streit, Wolfgang Ziegler, Michael Alexander, and Stephen Childs, editors. *Euro-Par 2007 Workshops: Parallel Processing, HPPC 2007, UNICORE Summit 2007, and VHPC 2007, Rennes, France, August 28-31, 2007, Revised Selected Papers*, volume 4854 of *Lecture Notes in Computer Science*. Springer, 2008.

[23] Don Box, Erik Christensen, Francisco Curbera, Donald Ferguson, Jeffrey Frey, Marc Hadley, Chris Kaler, David Langworthy, Frank Leymann, Brad Lovering, Steve Lucco, Steve Millet, Nirmal Mukhi, Mark Nottingham, David Orchard, John Shewchuk, Eugène Sindambiwe, Tony Storey, Sanjiva Weerawarana, and Steve Winkler. Web Services Addressing (WS-Addressing). Online `http://www.w3.org/Submission/ws-addressing/`, August 2004. W3C Member Submission.

[24] Andre Brinkmann, Stefan Gudenkauf, Wilhelm Hasselbring, André Höing, Odej Kao, Holger Karl, Holger Nitsche, and Guido Scherp. Employing WS-BPEL Design Patterns for Grid Service Orchestration using a Standard WS-BPEL Engine and a Grid Middleware. In *The 8th Cracow Grid Workshop*, pages 103 – 110, Cracow, Poland, March 2009. Academic Computer Center CYFRONET AGH.

[25] Rajkumar Buyya, Suraj Pandey, and Christian Vecchiola. Cloudbus Toolkit for Market-Oriented Cloud Computing. In *CloudCom '09: Proceedings of the 1st International Conference on Cloud Computing*, pages 24–44, Berlin, Heidelberg, 2009. Springer-Verlag.

[26] Rajkumar Buyya, Suraj Pandey, and Christian Vecchiola. Cloudbus Toolkit for Market-Oriented Cloud Computing. In Jaatun et al. [71], pages 24–44.

[27] Rajkumar Buyya, Chee S. Yeo, and Srikumar Venugopal. Market-Oriented Cloud Computing: Vision, Hype, and Reality for Delivering IT Services as Computing Utilities. In *HPCC '08: Proceedings of the 2008 10th IEEE International Conference on High Performance Computing and Communications*, pages 5–13, Washington, DC, USA, 2008. IEEE Computer Society.

[28] Kuo-Ming Chao, Muhammad Younas, Nathan Griffiths, Irfan Awan, Rachid Anane, and C-F Tsai. Analysis of Grid Service Composition with BPEL4WS. In *Proceedings of the 18th International Conference on Advanced Information Networking and Application (AINA'04)*, volume 01, page 284, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[29] David Chappell. *Enterprise Service Bus*. O'Reilly Media, Inc., 2004.

[30] Erik Christensen, Francisco Curbera, Greg Meredith, and Sanjiva Weerawarana. Web Service Definition Language (WSDL). Technical report, World Wide Web Consortium, March 2001.

[31] James Clark and Steve DeRose. XML Path Language (XPath). `http://www.w3.org/TR/xpath/`, Nov 1999.

[32] Workflow Management Coalition. Workflow Management Coalition Workflow Standard - Interoperability Wf-XML Binding. online `http://www.wfmc.org/standards/docs/Wf-XML-11.pdf` (last visited 23.06.2010), Nov 2001. Specification.

[33] OASIS Security Services Technical Committee. Security Assertion Markup Language (SAML). `http://docs.oasis-open.org/security/saml/v2.0/saml-2.0-os.zip`, March 2005.

[34] Mache Creeger. CTO Roundtable: Cloud Computing. *Commun. ACM*, 52(8):50–56, 2009.

[35] Francisco Curbera, Matthew J. Duftler, Rania Khalaf, and Douglas Lovell. Bite: Workflow Composition for the Web. In Bernd J. Krämer, Kwei-Jay Lin, and Priya Narasimhan, editors, *ICSOC*, volume 4749 of *Lecture Notes in Computer Science*, pages 94–106. Springer, 2007.

[36] Ewa Deelman, Gurmeet Singh, Mei-Hui Su, James Blythe, Yolanda Gil, Carl Kesselman, Gaurang Mehta, Karan Vahi, G. Bruce Berriman, John Good, Anastasia C. Laity, Joseph C. Jacob, and Daniel S. Katz. Pegasus: A framework for mapping complex scientific workflows onto distributed systems. *Scientific Programming*, 13(3):219–237, 2005.

[37] Tim Dornemann, Ernst Juhnke, and Bernd Freisleben. On-Demand Resource Provisioning for BPEL Workflows Using Amazon's Elastic Compute Cloud. In *CCGRID '09: Proceedings of the 2009 9th IEEE/ACM International Symposium on Cluster Computing and the Grid*, pages 140–147, Washington, DC, USA, 2009. IEEE Computer Society.

[38] T. Dörnemann, T. Friese, S. Herdt, E. Juhnke, and B. Freisleben. Grid Workflow Modelling Using Grid-Specific BPEL Extensions. German e-Sience Conference 2007, May 2007.

[39] Wolfgang Emmerich, Ben Butchart, Liang Chen, Bruno Wassermann, and Sarah L. Price. Grid Service Orchestration using the Business Process Execution Language (BPEL). *Journal of Grid Computing*, 3:283–304, 2005.

[40] Sharanya Eswaran, David Del Vecchio, Glenn Wasson, and Marty Humphrey. Adapting and Evaluating Commercial Workflow Engines for e-Science. In *E-SCIENCE '06: Proceedings of the Second IEEE International Conference on e-Science and Grid Computing*, page 20, Washington, DC, USA, 2006. IEEE Computer Society.

[41] OASIS eXtensible Access Control Markup Language Technical Committee. eXtensible Assertion Markup Language (XACML). `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=xacml`, July 2003.

[42] Onyeka Ezenwoye, S. Masoud Sadjadi, Ariel Cary, and Michael Robinson. Orchestrating WSRF-based Grid Services. Technical report, School of Computing and Information Sciences, Florida International University, April 2007.

[43] Arash Faroughi, Roozbeh Faroughi, Philipp Wieder, and Wolfgang Ziegler. Attributes and VOs: Extending the UNICORE Authorisation Capabilities. In Bougé et al. [22], pages 121–130.

[44] Christopher Ferris and David Langworthy. Web Services Reliable Messaging Protocol. (online) `http://download.boulder.ibm.com/ibmdl/pub/software/dw/specs/ws-rm/ws-reliablemessaging200502.pdf` (last visited 27.05.2010), Februrary 2005.

[45] Roy Thomas Fielding. *REST: Architectural Styles and the Design of Network-based Software Architectures*. Doctoral dissertation, University of California, Irvine, 2000.

[46] I. Foster, A. Grimshaw, P. Lane, W. Lee, M. Morgan, S. Newhouse, S. Pickles, D. Pulsipher, C. Smith, and M. Theimer. OGSA Basic Execution Service. online `http://www.ogf.org/documents/GFD.108.pdf` (last visited 23.06.2010), Nov 2008. Grid Final Document (GFD).

[47] I. Foster, C. Kesselman, J. Nick, and S. Tuecke. The Physiology of the Grid: An Open Grid Services Architecture for Distributed Systems Integration. http://www.globus.org/alliance/publications/papers/ogsa.pdf, 2002.

[48] Ian Foster, Carl Kesselman, Jeffrey M. Nick, and Steven Tuecke. Grid Services for Distributed System Integration. *Computer*, 35:37–46, 2002.

[49] Ian Foster, Carl Kesselman, and Steven Tuecke. The Anatomy of the Grid: Enabling Scalable Virtual Organizations. *International Jounral of Supercomputer Applications*, 15(3), 2001.

[50] Orlin Genchev and John Galletly. XPDL: bringing business and software together - a case study. In *CompSysTech '09: Proceedings of the International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing*, pages 1–6, New York, NY, USA, 2009. ACM.

[51] Y. Gil, E. Deelman, M. Ellisman, T. Fahringer, G. Fox, D. Gannon, C. Goble, M. Livny, L. Moreau, and J. Myers. Examining the Challenges of Scientific Workflows. *Computer*, 40(12):24–32, Dec. 2007.

[52] Steve Graham, Anish Karmarkar, Jeff Mischkinsky, Ian Robinson, and Igor Sedukhin. Web Services Resource 1.2 (WS-Resource). (Online) `http://docs.oasis-open.org/wsrf/wsrf-ws_resource-1.2-spec-os.pdf`, April 2006. OASIS Standard.

[53] Steve Graham and Jem Treadwell. Web Services Resource Properties 1.2 (WS-ResourceProperties). (Online) `http://docs.oasis-open.org/wsrf/wsrf-ws_resource_properties-1.2-spec-os.pdf`, April 2006. OASIS Standard.

[54] Zhijie Guan, Francisco Hernandez, Purushotham Bangalore, Jeff Gray, Anthony Skjellum, Vijay Velusamy, and Yin Liu. Grid-Flow: a Grid-enabled scientific workflow system with a Petri-net-based interface: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1115–1140, 2006.

[55] S. Gudenkauf, G. Scherp, W. Hasselbrink, A. Höing, and O. Kao. Workflow Modeling for WS-BPEL-based Service Orchestration in SMEs. In Gregor Engels, Markus Luckey, Alexander Pretschner, and Ralf Reussner, editors, *Software Engineering 2010 – Workshops*, volume P-160 of *Lecture Notes in Informatics*, pages 185–192. GI, 2010.

[56] Stefan Gudenkauf, Wilhelm Hasselbring, Felix Heine, André Höing, Odej Kao, and Guido Scherp. A Software Architecture for Grid Utilisation in Business Workflows. In Martin Bichler, Thomas Hess, Helmut Krcmar, Ulrike Lechner, Florian Matthes, Arnold Picot, Benjamin Speitkamp, and Petra Wolf, editors, *Multikonferenz Wirtschaftsinformatik*, pages 91–102. GITO-Verlag, Berlin, 2008.

[57] Stefan Gudenkauf, Wilhelm Hasselbring, Felix Heine, André Höing, Odej Kao, and Guido Scherp. BIS-Grid: Business Workflows for the Grid. In *The 7th Cracow Grid Workshop*, pages 86–93. Academic Computer Center CYFRONET AGH, 2008.

[58] Stefan Gudenkauf, Wilhelm Hasselbring, André Höing, Guido Scherp, and Odej Kao. Workflow Service Extensions for UNICORE 6 - Utilising a Standard WS-BPEL Engine for Grid Service Orchestration. In E. César, M. Alexander, A. Streit, J.L. Traff, C. Cérin, A. Knüpfer, D. Kranzlmüller, and S. Jha, editors, *Euro-Par 2008 Workshops - Parallel Processing*, volume Lecture Notes in Computer Science of *5415*, pages 103–112, April 2009.

[59] Stefan Gudenkauf, Wilhelm Hasselbring, André Höing, Guido Scherp, and Odej Kao. Using UNICORE and WS-BPEL for Scientific Workflow Execution in Grid Environments. In *Euro-Par 2009 Workshops - Parallel Processing*, volume 6043 of *Lecture Notes in Computer Science*, pages 335–344, 2010.

[60] Martin Gudgin, Marc Hadley, Noah Mendelsohn, Yves Lafon, Jean-Jacques Moreau, Anish Karmarkar, and Henrik Frystyk Nielsen. SOAP Version 1.2 Part 1: Messaging Framework (Second Edition), April 2007. http://www.w3.org/TR/2007/REC-soap12-part1-20070427/.

[61] Thilina Gunarathne, Chathura Herath, Eran Chinthaka, and Suresh Marru. Experience with adapting a WS-BPEL runtime for eScience workflows. In *GCE '09: Proceedings of the 5th Grid Computing Environments Workshop*, pages 1–10, New York, NY, USA, 2009. ACM.

[62] Felix Heine, André Höing, Stefan Gudenkauf, Guido Scherp, Holger Nitsche, and Jens Lischka. BIS-Grid - Betriebliche Informationssysteme: Grid-basierte Integration und Orchestrierung – Deliverable 3.1: WS-BPEL Engine Specification. (Online) `https://bi.offis.de/bisgrid/tiki-download_file.php?fileId=270` (last visited 27.05.2010), Decembre 2007.

[63] André Höing, Stefan Gudenkauf, and Guido Scherp. BIS-Grid - Betriebliche Informationssysteme: Grid-basierte Integration und Orchestrierung – Deliverable 3.5: Documentation GT4 Interoperarbility. (Online) `https://bi.offis.de/bisgrid/tiki-download_file.php?fileId=352` (last visited 26.05.2010), April 2010.

[64] André Höing, Stefan Gudenkauf, Guido Scherp, Holger Nitsche, and Dirk Meister. BIS-Grid - Betriebliche Informationssysteme: Grid-basierte Integration und Orchestrierung – Deliverable 3.4: Documentation BIS-Grid Workflow Engine Prototype. (Online) `https://bi.offis.de/bisgrid/tiki-download_file.php?fileId=351` (last visited 26.05.2010), April 2010.

[65] Andre Höing, Guido Scherp, and Stefan Gudenkauf. The BIS-Grid Engine: An Orchestration as a Service Infrastructure. *International Journal of Computing*, 8(3):96–104, December 2009.

[66] Christina Hoffa, Gaurang Mehta, Tim Freeman, Ewa Deelman, Kate Keahey, Bruce Berriman, and John Good. On the Use of Cloud Computing for Scientific Workflows. *eScience, IEEE International Conference on*, 0:640–645, 2008.

[67] Andreas Hoheisel. Grid Workflow Execution Service Dynamic and Interactive Execution and Visualization of Distributed Workflows. In *Proceedings of the Cracow Grid Workshop 2006, Cracow*, 2007.

[68] André Höing, Guido Scherp, and Stefan Gudenkauf. BIS-Grid - Betriebliche Informationssysteme: Grid-basierte Integration und Orchestrierung – Deliverable 2.1: Catalogue of WS-BPEL Design Patterns. (Online) `https://bi.offis.de/bisgrid/tiki-download_file.php?fileId=269` (last visited 27.05.2010), August 2008.

[69] André Höing, Guido Scherp, Stefan Gudenkauf, Dirk Meister, and André Brinkmann. An Orchestration as a Service Infrastructure Using Grid Technologies and WS-BPEL. In *ICSOC-ServiceWave '09: Proceedings of the 7th International Joint Conference on Service-Oriented Computing*, volume Volume 5900/2009 of *301–315*. Springer- Verlag Berlin / Heidelberg, Nov 2009.

[70] Internet2 Initiative. Shibboleth. Online `http://shibboleth.internet2.edu/` (last visited 23.04.2010).

[71] Martin Gilje Jaatun, Gansen Zhao, and Chunming Rong, editors. *Cloud Computing, First International Conference, CloudCom 2009, Beijing, China, December 1-4, 2009. Proceedings*, volume 5931 of *Lecture Notes in Computer Science*. Springer, 2009.

[72] Michael C. Jaeger, Gregor Rojec-Goldmann, and Gero Muhl. QoS Aggregation in Web Service Compositions. In *EEE '05: Proceedings of the 2005 IEEE International Conference on e-Technology, e-Commerce and e-Service (EEE'05) on e-Technology, e-Commerce and e-Service*, pages 181–185, Washington, DC, USA, 2005. IEEE Computer Society.

[73] I. Janciak, C. Kloner, and P. Brezany. Workflow enactment engine for WSRF-compliant services orchestration. In *GRID '08: Proceedings of the 2008 9th IEEE/ACM International Conference on Grid Computing*, pages 1–8, Washington, DC, USA, 2008. IEEE Computer Society.

[74] Niels Joncheere, Wim Vanderperren, and Ragnhild Van Der Straeten. Requirements for a Workflow System for Grid Service Composition. In Johann Eder and Schahram Dustdar, editors, *Business Process Management Workshops*, volume 4103 of *Lecture Notes in Computer Science*, pages 365–374. Springer, 2006.

[75] Diane Jordan and John Evdemon. Web Services Business Process Execution Language Version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, April 2007.

[76] Ernst Juhnke, Tim Dornemann, and Bernd Freisleben. Fault-Tolerant BPEL Workflow Execution via Cloud-Aware Recovery Policies. *Software Engineering and Advanced Applications, Euromicro Conference*, 0:31–38, 2009.

[77] Jae-Yoon Jung, Hoontae Kim, and Suk-Ho Kang. Standards-based approaches to B2B workflow integration. *Computers & Industrial Engineering*, 51(2):321 – 334, 2006. Special Issue: Logistics and Supply Chain Management, Selected Papers from The 33rd. ICC&IE.

[78] Matjaz B. Juric. EAI and Web Services. In *eAI Journal*, pages 31–35, 2002.

[79] Matjaz B. Juric. *Business Process Execution Language for Web Services BPEL and BPEL4WS 2nd Edition*. Packt Publishing, 2006.

[80] Carl Kesselman and Ian Foster. *The Grid: Blueprint for a New Computing Infrastructure*. Morgan Kaufmann Publishers, November 1998.

[81] Carl Kesselman and Ian Foster. *The Grid. Blueprint for a New Computing Infrastructure*. Morgan Kaufmann, 2. a. edition, December 2003.

[82] E. Kieselstein, S. Zimmer, J. Kieselstein, F. Swoboda, P. Hillebrand, and E. Mauersberger. BIS-Grid - Betriebliche Informationssysteme: Grid-basierte Integration und Orchestrierung – Deliverable 4.7: Dokumentation der auf BIS-Grid basierenden Softwarelösung für das Anwendungsszenatrio KIESELSTEIN. (confidential), April 2010.

[83] Sriram Krishnan, Patrick Wagstrom, and Gregor von Laszewski. GSFL: A Workflow Framework for Grid Services. Technical report, ARGONNE NATIONAL LABORATORY, 9700 S. CASS AVENUE, ARGONNE, IL 60439, 2002.

[84] Roland Kübert, Hai-Lang Thai, and Axel Tenschert. A SOAP performance comparison of different WSRF implementations. In *MEDES '09: Proceedings of the International Conference on Management of Emergent Digital EcoSystems*, pages 269–273, New York, NY, USA, 2009. ACM.

[85] Kelvin Lawrence and Chris Kaler. Web Services Security: 4 SOAP Message Security 1.1. online `http://www.oasis-open.org/committees/download.php/16790/wss-v1.1-spec-os-SOAPMessageSecurity.pdf`, Februrary 2006.

[86] Kelvin Lawrence and Chris Kaler. WS-SecureConversation. online `http://docs.oasis-open.org/ws-sx/ws-secureconversation/v1.4/os/ws-secureconversation-1.4-spec-os.pdf`, February 2009.

[87] Stefan Leugers, Herbert Nase, and Manfred Neugebauer. BIS-Grid - Betriebliche Informationssysteme: Grid-basierte Integration und Orchestrierung – Deliverable 4.3: Dokumentation der auf BIS-Grid basierenden Softwarelösung für das Anwendungsszenatrio CeWe Color. (confidential), April 2010.

[88] Frank Leymann. Web Services Flow Language. http://www.itee.uq.edu.au/ infs7201/Assessments/AssignmentFL.pdf, May 2001.

[89] Frank Leymann. Choreography for the Grid: towards fitting BPEL to the resource framework: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1201–1217, 2006.

[90] Frank Leymann, Dieter Roller, and Satish Thatte. Goals of the BPEL4WS Specification. Online `http://xml.coverpages.org/BPEL4WS-DesignGoals.pdf` (last visited 09.05.2010).

[91] Lily Liu and Sam Meder. Web Services Base Faults 1.2 (WS-BaseFaults). (Online) `http://docs.oasis-open.org/wsrf/wsrf-ws_base_faults-1.2-spec-os.pdf`, April 2006. OASIS Standard.

[92] Bertram Ludäscher, Ilkay Altintas, Chad Berkley, Dan Higgins, Efrat Jaeger, Matthew Jones, Edward A. Lee, Jing Tao, and Yang Zhao. Scientific workflow management and the Kepler system. *Concurr. Comput. : Pract. Exper.*, 18(10):1039–1065, 2006.

[93] Bertram Ludäscher, Mathias Weske, Timothy McPhillips, and Shawn Bowers. Scientific Workflows: Business as Usual? In Umeshwar Dayal, Johann Eder, Jana Koehler, and Hajo Reijers, editors, *7th Intl. Conf. on Business Process Management (BPM)*, LNCS 5701, Ulm, Germany, 2009.

[94] Ru-Yue Ma, Yong-Wei Wu, Xiang-Xu Meng, Shi-Jun Liu, and Li Pan. Grid-Enabled Workflow Management System Based On BPEL. *Int. J. High Perform. Comput. Appl.*, 22(3):238–249, 2008.

[95] Tom Maguire, David Snelling, and Tim Banks. Web Services Service Group 1.2 (WS-ServiceGroup). (Online) `http://docs.oasis-open.org/wsrf/wsrf-ws_service_group-1.2-spec-os.pdf`, April 2006. OASIS Standard.

[96] Lawrence Mandel. Describe REST Web services with WSDL 2.0. Technical report, IBM, May 2008. (online) `http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-restwsdl/ws-restwsdl-pdf.pdf` (last visited 20.05.2010).

[97] C. Matthiew, Ken Laskey, Francis McCabe, Peter F Brown, and Rebekah Matz. Reference Model for Service Oriented Architecture 1.0. Technical report, OASIS, October 2006.

[98] Robert McGill, John W. Tukey, and Wayne A. Larsen. Variations of Box Plots. *The American Statistician*, 32(1):12–16, 1978.

[99] Massimo Mecella, Barbara Pernici, Monica Rossi, and Andrea Testi. A Repository of Workflow Components for Cooperative e-Applications. In *in Proceedings of the 1st IFIP TC8 Working Conference on E-commerce/E-business*, 2001.

[100] Massimo Mecella, Francesco Parisi Presicce, and Barbara Pernici. Modeling E-service Orchestration through Petri Nets. In *in Proceedings of the 3rd VLDB International Workshop on Technologies for e-Services (VLDB-TES 2002), Hong Kong, Hong Kong SAR*, pages 38–47. Springer-Verlag, 2002.

[101] OASIS BPEL4PEOPLE Technical Committee. WS-BPEL Extension for People (BPEL4People). http://www.oasis-open.org/committees/download.php/27505/BPEL4Peopleontribution.pdf, June 2007.

[102] OASIS WSBPEL Technical Committee. Web Services Business Process Execution Language (WSBPEL) Primer. http://www.oasis-open.org/committees/download.php/23974/wsbpel-v2.0-primer.pdf, May 2007.

[103] OASIS WSRF Technical Committee. Web Services Resource Framework v1.2. `http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsrf`.

[104] Tom Oinn, Matthew Addis, Justin Ferris, Darren Marvin, Tim Carver, Matthew R. Pocock, and Anil Wipat. Taverna: A tool for the composition and enactment of bioinformatics workflows. *Bioinformatics*, 20:2004, 2004.

[105] OMG. Business Process Model and Notation (BPMN). http://www.omg.org/spec/BPMN/2.0/Beta1/PDF, August 2009. Version 2.0 - Beta 1.

[106] C. Peltz. Web services orchestration and choreography. *Computer*, 36(10):46–52, 2003.

[107] Lavanya Ramakrishnan, Daniel Nurmi, Anirban Mandal, Charles Koelbel, Dennis Gannon, T. Mark Huang, Yang-Seok Kee, Graziano Obertelli, Kiran Thyagaraja, Rich Wolski, Asim YarKhan, and Dmitri Zagorodnov. VGrADS: Enabling e-Science Workflows on Grids and Clouds with Fault Tolerance. In *SC'09 The International Conference for High Performance Computing, Networking, Storage and Analysis*, Portland, OR, 2009. (Accepted).

[108] Poornachandra Sarang, Ramesh Loganathan, Matjaz B. Juric, and Frank Jennings. *SOA Approach to Integration: XML, Web services, ESB, and BPEL in real-world SOA projects*. Packt Publishing, 2007.

[109] Bernd Schuller, Bastian Demuth, Hartmut Mix, Katharina Rasch, Mathilde Romberg, Sulev Sild, Uko Maran, Piotr Bala, Enrico del Grosso, Mosé Casalegno, Nadège Piclin, Marco Pintore, Wibke Sudholt, and Kim Baldridge. Chemomentum - UNICORE 6 Based Infrastructure for Complex Applications in Science and Technology. In Bougé et al. [22], pages 82–93.

[110] Satoshi Shirasuna, Aleksander Slominski, Liang Fang, and Dennis Gannon. Performance Comparison of Security Mechanisms for Grid Services. *Grid Computing, IEEE/ACM International Workshop on*, 0:360–364, 2004.

[111] Aleksander Slomiski. On using BPEL extensibility to implement OGSI and WSRF Grid workflows: Research Articles. *Concurr. Comput. : Pract. Exper.*, 18(10):1229–1241, 2006.

[112] Aleksander Slomiski. Adapting BPEL to Scientific Workflows. In Dennis B. Gannon Ian J. Taylor, Ewa Deelman and Matthew Shields, editors, *Workflows for e-Science*, pages 212–230, December, 2007. Springer London.

[113] Mirko Sonntag, Dimka Karastoyanova, and Frank Leymann. The Missing Features of Workflow Systems for Scientific Computations. In *Software Engineering 2010 – Workshops*, pages 209–216. Gesellschaft für Informatik e.V. (GI), Februar 2010.

[114] Latha Srinivasan and Tim Banks. Web Services Resource Lifetime 1.2. Online `http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf`, April 2006.

[115] Latha Srinivasan and Tim Banks. Web Services Resource Lifetime 1.2 (WS-ResourceLifetime). (Online) `http://docs.oasis-open.org/wsrf/wsrf-ws_resource_lifetime-1.2-spec-os.pdf`, April 2006. OASIS Standard.

[116] SWITCH. Description of the SLCS. Online `http://www.switch.ch/grid/slcs/about/about_long.html` (last visited 23.04.2010).

[117] Wei Tan, Liana Fong, and Norman Bobroff. BPEL4Job: A Fault-Handling Design for Job Flow Management. In *ICSOC '07: Proceedings of the 5th international conference on Service-Oriented Computing*, pages 27–42, Berlin, Heidelberg, 2007. Springer-Verlag.

[118] Koon Leai Larry Tann and Kenneth J. Turner. Orchestrating Grid Services using BPEL and Globus Toolkit 4. In *7th Annual Post Graduate Symposium on the Convergence of Telecommunications, Networking and Broadcasting*, June 2006.

[119] Ian Taylor, Matthew Shields, Ian Wang, and Roger Philp. Grid Enabling Applications Using Triana. In *Workshop on Grid Applications and Programming Tools*. Held in Conjunction with GGF8, 2003.

[120] Condor Team. DAGMan. Online `http://www.cs.wisc.edu/condor/dagman/` (last visited 12.05.2010).

[121] Douglas Thain, Todd Tannenbaum, and Miron Livny. Condor and the grid. In *Grid Computing: Making the Global Infrastructure a Reality*. John Wiley, 2003.

[122] Rajesh K. Thiagarajan, Amit K. Srivastava, Ashis K. Pujari, and Visweswar K. Bulusu. BPML: A Process Modeling Language for Dynamic Business Models. In *WECWIS '02: Proceedings of the Fourth IEEE International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems (WECWIS'02)*, page 239, Washington, DC, USA, 2002. IEEE Computer Society.

[123] S. Tilley, J. Gerdes, T. Hamilton, S. Huang, H. Müller, D. Smith, and K. Wong. On the business value and technical challenges of adopting web services. *J. Softw. Maint. Evol.*, 16(1-2):31–50, 2004.

[124] S. Tuecke, K. Czajkowski, I. Foster, J. Frey, S. Graham, C. Kesselman, Http://www. Ggf. Org/ogsi wg K. Czajkowski, P. Vanderbilt (eds.), I. Foster Anl, J. Frey Ibm, C. Kesselman Usc/isi, D. Snelling, Fujitsu Labs, and P. Vanderbilt Nasa. Open Grid Services Infrastructure (OGSI), 2003.

[125] W. M. P. van der Aalst and Ter. YAWL: yet another workflow language. *Information Systems*, 30(4):245–275, June 2005.

[126] Luis M. Vaquero, Luis Rodero-Merino, Juan Caceres, and Maik Lindner. A Break in the Clouds: Towards a Cloud Definition. *SIGCOMM Comput. Commun. Rev.*, 39(1):50–55, 2009.

[127] Asir S Vedamuthu, David Orchard, Frederick Hirsch, Maryann Hondo, Prasad Yendluri, Toufic Boubez, and Ümit Yalçinalp. Web Services Policy 1.5 - Framework. Online `http://www.w3.org/TR/ws-policy/` (last vistited 10.06.2010), September 2007. W3C Recommendation.

[128] Steve Vinoski. Java Business Integration. *IEEE Internet Computing*, 9:89–91, 2005.

[129] V. Welch. Globus Toolkit Version 4 Grid Security Infrastructure: A Standards Perspective. http://www.globus.org/toolkit/docs/4.0/security/GT4-GSI-Overview.pdf, 2005.

[130] Von Welch, Tom Barton, Kate Keahey, and Frank Siebenlist. Attributes, Anonymity, and Access: Shibboleth and Globus.Integration to Facilitate Grid Collaboration. In *Proc. Fourth Ann. Public Key Infrastructure R&D Workshop*, 2005.

[131] Tomasz Wiktor Wlodarczyk, Chunming Rong, and Kari Anne Haaland Thorsen. Industrial Cloud: Toward Inter-enterprise Integration. In Jaatun et al. [71], pages 460–471.

[132] L. Yousseff, M. Butrico, and D. Da Silva. Toward a Unified Ontology of Cloud Computing. In *Grid Computing Environments Workshop, 2008. GCE '08*, November 2008.

[133] Jia Yu and Rajkumar Buyya. A Novel Architecture for Realizing Grid Workflow using Tuple Spaces. In *GRID '04: Proceedings of the 5th IEEE/ACM International Workshop on Grid Computing*, pages 119–128, Washington, DC, USA, 2004. IEEE Computer Society.

[134] Tao Yu and Kwei jay Lin. Service selection algorithms for composing complex services with multiple qos constraints. In *In: ICSOC 05: 3rd Int. Conf. on Service Oriented Computing*, pages 130–143, 2005.

[135] Zhili Zhao, Ruisheng Zhang, Jiazao Lin, Ying Chen, Huajian Zhang, and Lian Li. An Improved Visual BPEL-Based Environment for Scientific Workflow. In *GCC '08: Proceedings of the 2008 Seventh International Conference on Grid and Cooperative Computing*, pages 435–441, Washington, DC, USA, 2008. IEEE Computer Society.