Martin Beckmann, Andreas Vogelsang

What is a good textual representation of activity diagrams in requirements documents?

Suggested Citation
Beckmann, Martin; Vogelsang, Andreas: What is a good textual representation of activity diagrams in requirements documents? - In: Requirements Engineering Conference Workshops (REW), 2017 IEEE 25th International. - New York: IEEE, 2017. ISBN: 978-1-5386-3488-2. - S. 56-63. - DOI: 10.1109/REW.2017.19. *(Postprint version is cited, page numbers may differ.)*

What is a Good Textual Representation of Activity Diagrams in Requirements Documents?

Martin Beckmann
Technische Universität Berlin, Germany
martin.beckmann@tu-berlin.de

Andreas Vogelsang
Technische Universität Berlin, Germany
andreas.vogelsang@tu-berlin.de

***Abstract*—The use of graphical models has become a widely adopted approach to specify requirements of complex systems. Still, in practice, graphical models are often accompanied by textual descriptions to provide more detail, because of legal considerations, and to enable stakeholders with different backgrounds to understand a requirements document. One of our industry partners (Daimler AG) uses activity diagrams to specify vehicle functions in combination with a textual representation thereof in their requirements documents. Since graphical and textual representations serve different purposes, it is not obvious how textual representations of activity diagrams should be structured. In this paper, we present different textual representations of activity diagrams for use in requirements documents. The representation currently in use is presented as well as four alternatives. For each representation, we discuss advantages and disadvantages. To evaluate the representations, we asked five stakeholders of one system to create a preference ranking of the representations. The resulting ranking showed that the currently used representation is not considered to be the best possible option. The stakeholders' favorite textual representation emphasizes structural similarity with the activity diagram, which however does not resemble the diagram's structure exactly.**

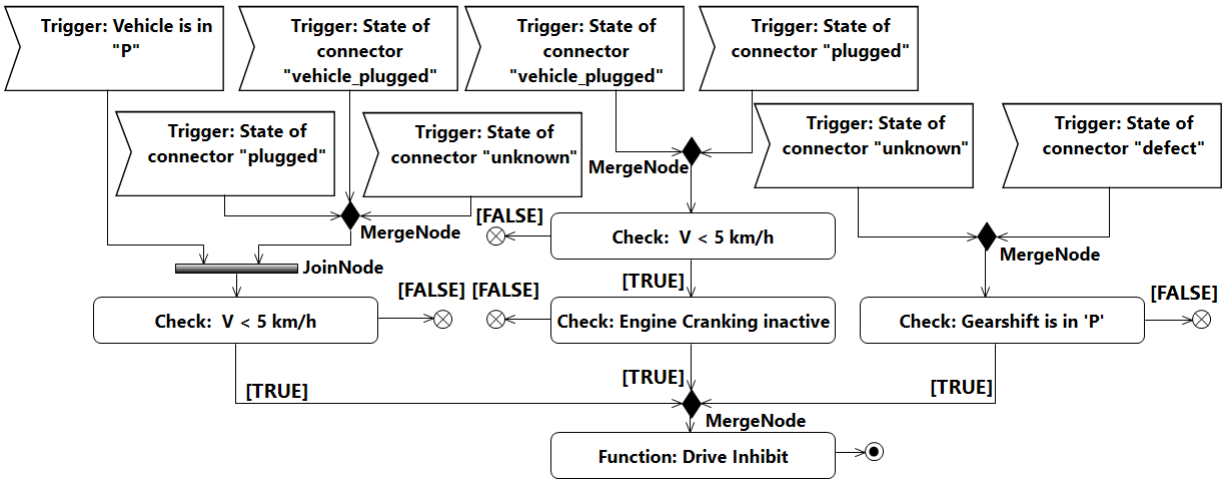
I. Introduction

Complex software systems, which, for example, can be found in distributed embedded systems in automotive electronics, require model-based and system-oriented development approaches [1]. Using graphical models for specification manages complexity and improves reusability and analytical capabilities [2], [3]. Although graphical models provide suitable means to specify and understand dependencies and procedural behavior of a system, in industry they are usually accompanied by textual representations. Related work has shown the need for a continuous systems engineering environment, where referring or constitutive documents are essential to work on complex software systems [4]. Also, the combined use of graphical diagrams and textual descriptions is considered beneficial for the requirements management process [5], [6]. In addition, for industrial applications, model exchange for graphical models is still not properly supported by tool vendors. As a result, the handover between manufacturers and suppliers is still performed based on textual documents. This is especially important since these textual documents often serve as the basis for legal considerations between the contractors [6], [7]. Another reason why graphical models are usually accompanied by textual descriptions is the background of different stakeholders—not everyone is capable of understanding the graphical models [8], [9]. Thus, the information contained in a model needs to be written in words to be appropriately reviewable [10]. A textual representation also allows making implicit information of a model explicit [7] and hence more accessible.

The Daimler AG uses a specification approach, where, as a first step, a UML activity diagram [11] is created for each function of a vehicle system to describe the function's activation and deactivation by triggers and conditions. This kind of description is also known in literature to formulate textual natural language requirements [12]. A textual representation of each activity diagram along with the diagram itself is then transferred into a requirements document for everyone to understand and for ongoing development. The transformation of the activity diagrams into textual representations is done manually (i.e., not generated by a tool). This manual process is error-prone and labor intensive on the one hand [13] but, on the other hand, provides some flexibility to create a textual representation that best fits its purpose. Since graphical and textual representations serve different purposes, it is not obvious how textual representations of activity diagrams should be structured.

In this paper, we present the currently used textual representation and four alternative textual representations of the sort of activity diagrams our industry partner uses. We explain what aspects of an activity diagram the textual representations describe and discuss their advantages and disadvantages. To assess which of the textual representations is most suitable to describe a function's behavior, we asked five stakeholders of one particular system to rank these representations. Although the currently used representation is among the best in the ranking, all stakeholders but one preferred a different textual representation. In comparison this textual representation has a more refined structure, which is realized by using additional textual objects.

The paper is structured as following: The next section presents how Daimler uses activity diagrams in their specification process and shows the currently used textual representation. Section III presents related work on the topic of deriving textual requirements documents or parts thereof from graphical models. In the fourth section, we present alternative textual representations and discuss their advantages and disadvantages. In Section V, we explain the details of our evaluation and present its results. The last section concludes this work.



(a) Activity diagram of the function *Drive Inhibit*

ID	Text	Level	Type
1000	**Drive Inhibit**	2	Function
1236	State of connector "unknown" OR State of connector "defect" OR	3	Trigger
1237	Vehicle Gear Selector is in position "P"	4	Check
1111	State of connector "plugged on vehicle side" ("VEH_PLUGGED") OR "plugged on vehicle and EVSE side" ("PLUGGED"). OR	3	Trigger
1112	Vehicle velocity is below 5 km/h AND	4	Check
1113	Engine Cranking inactive	4	Check
1114	Vehicle Gear selector is in position "P" OR	3	Trigger
1232	Vehicle velocity is below 5 km/h	4	Check
1233	State of connector "plugged on vehicle side" OR State of connector "plugged on vehicle and EVSE side" OR State of connector "unknown" AND	3	Trigger
1238	Vehicle velocity is below 5 km/h	4	Check

(b) Textual representation of the function *Drive Inhibit*

Fig. 1: Activity diagram and the specification text of a function

II. Background

Daimler uses UML activity diagrams to specify functions of a system. Creating an activity diagram is the first step of specifying a new function. Activity diagrams are used to provide an early overview of the desired function behavior with a special focus on the function's activation, execution conditions, functional paths, and deactivation. The information contained in the activity diagram as well as the activity diagram itself is then transferred to a textual requirements document. This transfer is necessary since this textual requirements document is the central artifact for further development (e.g., it is the basis for creating test cases or handing over requirements to suppliers). Besides, the textual document contains additional and more detailed information as well as statements about its context, which relates this approach to Literate Modelling [8].

Fig. 1 shows an exemplar specification as we have found it at Daimler. The example consists of an activity diagram and its textual representation in the requirements document.

Fig. 1a shows the activity diagram of the function *Drive Inhibit*. The actual behavior of the activated function is described in the *Action* node labeled with *Drive Inhibit* (bottom of the diagram). The function's activation is described by a combination of triggers and checks for conditions. For triggers, the *AcceptEventAction* element is used. The checks are modeled as *Action* elements. If the condition of a check is not fulfilled, the flow ends (*FlowFinal*). The triggers and checks are connected by *ControlNodes* such as *JoinNodes* and *MergeNodes*. *JoinNodes* act as synchronization points and can be interpreted as AND operators in terms of propositional logic. *MergeNodes* represent OR operators. Once the actual functionality of the function is executed, *ActivityFinal* elements designate the end of an activity.

The currently used textual representation of the activity is shown in Fig. 1b. Each row in the document represents an object, which is described by a set of attributes (columns). The *ID* attribute contains a unique identifier of the object. The *Text* attribute is a textual description of the object and is supposed to be equal to the text of the corresponding element in the activity diagram. The *Level* is an attribute to structure the document hierarchically. It is derived from the structure of the activity diagram. The *Type* attribute of each text object is supposed to be equal to the type of its corresponding element in the diagram. These attributes are needed to display the relevant information of the activity diagram in the requirements document. Besides

the given attributes, the textual document contains additional attributes used for further development, which are not shown in our example.

Propositional logic operators such as OR and AND are used as strings in the *Text* attributes of the objects to describe the logic statements of the activity diagram. The operators at the end of an object's text connect the object with the following object on the same level of the document hierarchy. For instance, in Fig. 1b, the object with *ID 1236* is connected via an OR with the object with *ID 1111* because it is the next object on the same hierarchical level. Besides the description of propositional logic relations, the different levels of the document are used to indicate the order of the elements within the activity diagram. For example, the check *Vehicle Gear selector is in position "P"* (*ID 1237*) is executed after one of the triggers contained in the object with the *ID 1236* occurred. Hence, it appears one level below. There might be more than one check associated with a set of triggers (e.g., text object with *ID 1112* and *ID 1113*). In this case, they appear on the same level.

The current textual representation puts a focus on the logic relations between the individual elements by adding them explicitly in their texts. By using different levels of the document, it is possible to see which checks belong to which group of triggers. This also reflects different paths through the diagram and thus focuses on necessary elements in the activity that lead to the function's execution.

The currently employed manual transition process from activity diagrams to textual representations poses the risk of introducing a number of inconsistency issues. A number of these issues can be seen in Fig. 1. In a former study, we investigated these issues in detail [13]. While some of these issues can be resolved [14], some of the other issues originate from the currently used textual representation. Alternative representations might mitigate the consequences or resolve these issues altogether.

III. Related Work

Creating human-usable textual notations of MOF models has been addressed by the OMG HUTN Specification [15]. Since the notation is generically applicable to all MOF models, it does not take into account certain aspects of activity diagrams or behavioral models sufficiently. The focus on certain features of a diagram type is necessary because generic notations are harder to understand and thus not suitable to derive understandable requirements.

Deriving requirements and structures for requirements documents from graphical models is an established approach [16]. Especially UML/SysML diagrams have received attention. Class diagrams, as the most used UML diagram type [17] of the UML, have been used to generate natural language specifications [18]. Robinson-Mallett shows how Statecharts and Block Diagrams can be used to create a structure for a requirements document [19]. Berenbach introduces an algorithm that derives a structure for a requirements document from use case diagrams [20]. Using activities as a source for requirements has already been addressed by Drusinsky [21], however, only for UML-1. Additionally, only the generation of requirements is addressed but not the creation of a requirements document structure. Besides creating textual representations from activities in the context of requirements engineering, there has been work on rendering all aspects of an activity as a text [22]. Whether this approach is applicable in requirements engineering is also part of this paper. In addition the Action Language for Foundational UML (Alf) can also be used as an alternative textual notation of a model [23].

IV. Textual Representations of Activities

Just like models themselves, their textual representations may emphasize specific aspects of a model. Depending on the purpose, a good textual representation should focus on different aspects of the model. For activity diagrams these are (but not limited to): propositional logic relations, order of execution, number of executions of actions, asynchronous events, possible paths in an activity, and parallel processing of actions. In the following, we present four textual representations that may be used as alternatives to the currently used (original) style of transforming activity diagrams into textual representations at Daimler. As the description of propositional logic relations is very important for the Daimler specification approach, there is an emphasis on propositional logic in all of the representations. Although these textual representations may potentially be processed by stakeholders without in-depth knowledge of activity diagram notations, it can still be assumed that the stakeholders have some general background on concepts and models used in computer science.

A. Grouping

In Fig. 2, an alternative textual representation of the activity in Fig. 1a is displayed. We call this textual representation *grouping* because it introduces additional *GROUP* elements in the text. These elements are used to group elements that are connected by the same logic connector. The elements belonging to one group are placed one level below the *GROUP* element. The curly brackets are used to make it easier to perceive, which elements belong together. Since the relations between the elements are already achieved by using different levels in the document, the brackets are optional. The groups themselves can be connected to other groups or elements via logical operators. In contrast to the original representation, the grouping avoids the repetition of elements and ensures that the propositional logic of the activity is correctly reproduced in the requirements document. Another advantage is that the representation still resembles the structure of the activity diagram as the paths are still recognizable. Besides, its structure is closely related to the original representation and thus it is easy for stakeholders at Daimler to adapt to this new textual representation.

We additionally introduce a *THEN* operator that describes that *Actions* are executed consecutively. This means, that every *Action* only starts executing, when its predecessors have successfully finished their executions. This way, it is also possible to represent the order of executions of actions in the paths of the activity. An AND operator on the other hand represents a

Text	Level	Type
Drive Inhibit	2	Function
GROUP {	3	-
GROUP {	4	-
Vehicle is in "P" AND	5	Trigger
GROUP {	5	-
State of connector "plugged" OR	6	Trigger
State of connector "vehicle_plugged" OR	6	Trigger
State of connector "unknown"	6	Trigger
}	5	-
} THEN	4	-
V < 5 km/h	4	Check
} OR	3	-
GROUP {	3	-
GROUP {	4	-
State of connector "plugged" OR	5	Trigger
State of connector "vehicle_plugged"	5	Trigger
} THEN	4	-
V < 5 km/h THEN	4	Check
Engine Cranking inactive	4	Check
} OR	3	-
GROUP {	3	-
GROUP {	4	-
State of connector "defect" OR	5	Trigger
State of connector "unknown"	5	Trigger
} THEN	4	-
Gearshift is in "P"	4	Check
}	3	-

Fig. 2: Grouping Representation

Text	Level	Type
Drive Inhibit	2	Function
GROUP OR	3	-
State of connector "plugged"	4	Trigger
Vehicle is in "P"	4	Trigger
V < 5 km/h	4	Check
GROUP OR	3	-
State of connector "vehicle_plugged"	4	Trigger
Vehicle is in "P"	4	Trigger
V < 5 km/h	4	Check
GROUP OR	3	-
State of connector "unknown"	4	Trigger
Vehicle is in "P"	4	Trigger
V < 5 km/h	4	Check
GROUP OR	3	-
State of connector "vehicle_plugged"	4	Trigger
V < 5 km/h	4	Check
Engine Cranking inactive	4	Check
GROUP OR	3	-
State of connector "plugged"	4	Trigger
V < 5 km/h	4	Check
Engine Cranking inactive	4	Check
GROUP OR	3	-
State of connector "unknown"	4	Trigger
Gearshift is in "P"	4	Check
GROUP	3	-
State of connector "defect"	4	Trigger
Gearshift is in "P"	4	Check

Fig. 3: Normal Form Representation

JoinNode and thus indicates that all connected elements can be executed independently. A drawback of this representation is that it needs additional grouping elements to correctly describe the activity's structure. As these elements are not requirements per se, the description becomes longer and also needs additional levels. This may impede the understandability of the function execution.

B. Normal Form

In Fig. 3, another textual representation of the activity in Fig. 1a is shown. We call this textual representation *normal form* because it represents a disjunctive normal form of the propositional logic statement underlying the activity diagram. Therefore, this presentation solely describes the aspect of propositional logic in activities and refrains from describing the execution order or parallel processing. In contrast to the original representation, this ensures that the propositional logic of the activity is correctly reproduced. Additionally, the conversion into a normal form simplifies the representation of the underlying propositional formula.

Similar to the *grouping* representation, elements are structured into groups by inserting an object in the text, denoted with the string *GROUP*. All elements of a group are placed one level below that *GROUP* element. The elements of a group are logically connected by an AND operator (omitted in the example), while all the groups are connected by an OR operator.

Due to the OR connections between groups, an execution of all elements in any group results in the activation of the whole function. Therefore, this representation emphasizes distinct combinations of elements that cause a function's activation. In the normal form, only two hierarchical levels are needed to display the representation in the requirements document.

One of the disadvantages of this representation is that its structure does not resemble the structure of the activity diagram. Additionally, the generation of the normal form suppresses the order of execution of the elements. Hence, the sequence, in which the elements need to be executed, is not part of the representation. This drawback may be mitigated by using the order of appearance beneath a grouping element as an indicator for execution sequences. However, a group might also contain elements that are independently executable; an information that gets lost if order of appearance is interpreted as execution order. Additional structural elements would be necessary to express the independence of certain elements.

C. Tree

In Fig. 4, a third textual representation of the activity in Fig. 1a is displayed. We call this textual representation *tree* because it uses the hierarchical document structure to display the expression tree [24] of the propositional logic statements in the diagram. In this representation, the logic operators AND and OR are distinct objects in the requirements document. All elements that are one level below an operator are logically connected by that operator. An operator element might contain further operators as elements. This ensures that logic relations between the elements of the diagram are correctly reproduced in the textual representation. The tree representation reflects the logical statement as it appears in the diagram (i.e., no simplifications or transformations are done).

This representation has the drawback that its structure does not resemble the structure of the original activity diagram,

Text	Level	Type
Drive Inhibit	2	Function
OR	3	-
AND	4	-
Vehicle is in "P"	5	Trigger
OR	5	-
State of connector "plugged"	6	Trigger
State of connector "vehicle_plugged"	6	Trigger
State of connector "unknown"	6	Trigger
V < 5 km/h	5	Check
AND	4	-
OR	5	-
State of connector "plugged"	6	Trigger
State of connector "vehicle_plugged"	6	Trigger
V < 5 km/h	5	Check
Engine Cranking inactive	5	Check
AND	4	-
OR	5	-
State of connector "defect"	6	Trigger
State of connector "unknown"	6	Trigger
Gearshift is in "P"	5	Check

Fig. 4: Tree Representation

Text	Level	Type
Drive Inhibit	2	Function
>Trigger: Vehicle is in "P"] -> <JoinNode *jn1> -> (Check: V < 5 km/h) -> <MergeNode *mn1> -> (Function: Drive Inhibit) -> <ActivityFinal> >Trigger: State of connector "vehicle_plugged"] -> <MergeNode *mn2> -> <*jn1> >Trigger: State of connector "plugged"] -> <*mn2> >Trigger: State of connector "unknown"] -> <*mn2> >Trigger: State of connector "vehicle_plugged"] -> <MergeNode *mn3> -> (Check: V < 5 km/h) -> (Check: Engine Cranking inactive) -> <*mn1> >Trigger: State of connector "plugged"] -> <MergeNode *mn3> >Trigger: State of connector "defect"] -> <MergeNode *mn4> -> (Check: Gearshift is in "P") -> <*mn1> >Trigger: State of connector "unknown"] -> <MergeNode *mn4>	3	-

Fig. 5: Exact Equivalent Representation

which makes it harder to recognize the relations between the two artifacts. Also, a basic understanding of expression trees may be necessary to comprehend the connections between the elements. As tree structures are suitable to express all kinds of formulas, it is also possible to add more operators aside from the propositional logic operators (e.g., including a *THEN* operator for expressing sequences of actions).

D. Exact Equivalent Representation

This textual representation of activities was suggested by Flater et al. [22]. Their idea is to convert the complete activity into human-readable text. We call this *Exact Equivalent Representation* because it maps each visual element of an activity to a textual string. Hence, the resulting text includes every aspect of the original activity. Elements of activities are represented by symbols such as parenthesis (as actions nodes) and square brackets (as object nodes). *ActivityEdges* are depicted by ASCII arrows (<- and ->). Elements that are used multiple times contain the name of the elements followed by an asterisk and a variable name, which is used as reference. Since the referenced work did not include *AcceptEventActions*, we use a greater-than sign and a square bracket (> *ElementText]*) to denote these elements.

The resulting textual representation for the example in Fig. 1a is shown in Fig. 5. Since the textual notation is a serialization of the activity, the multiple levels of a hierarchically structured document are not necessary and the transformation is placed in a single entry. This representation has no loss of information besides the layout of the activity diagram. Hence, it is applicable to all sorts of activity diagrams. The authors also mention that using this textual representation instead of graphical representations does not require special tooling and reduces effort when implementing prototypes [22]. Not using special tooling, on the other hand, allows for the construction of invalid expressions.

A major drawback of this representation is that it is difficult to get a quick overview of the function and to grasp the relations between elements. In addition, stakeholders not familiar with activity diagrams have no advantage in understanding this type of text better than an activity diagram itself.

E. Other Possible Representations

In the context of the Daimler specification approach, parallel processing of actions is not relevant. Therefore, the presented representations do not incorporate this aspect specifically. However, OR operators, which are derived from *MergeNodes* in an activity, connect elements that are independently executable. The normal form representation may be complemented by additional key words that group elements that are independently executable. However, this makes the textual representation harder to understand because it requires an additional grouping object and thus an additional level in the document structure.

Expressing parallelism as text is also possible by using parbegin/parend or join/fork statements. Though, we doubt that these options satisfy the need that these statements are easier to understand than activities.

V. Evaluation

To assess how practitioners perceive the usefulness of the presented representations and to learn more about preferences they have, we performed a survey, in which we asked stakeholders of one particular system to create a ranking of the representations.

The survey consists of three parts: At first, the survey document presents an original activity diagram of a function of the system the participants are involved in. The function is the same as in Fig. 1a, however, we presented the activity diagram as it looks like in the tool used (Enterprise Architect) by the participants. The one in their tool and the one displayed in Fig. 1a differ slightly in terms of color and layout.

Following the activity diagram, the different textual representations of the diagram are presented. Again, they are presented as they would look like in the tool used (IBM DOORS) by the participants. Each representation is accompanied by an explaining text and a listing of its advantages and disadvantages.

The last part of the survey document contains a pairwise comparison of the textual representations. Since we examine

5 different textual representations, the pairwise comparison consists of 10 comparisons to cover all combinations. For each comparison, the participants had to provide which representation they perceive as more useful or whether both are equally useful. Each time a representation surpasses another representation it is accredited with 1 point, while the losing representation is accredited with -1 point. In case of a tie both representations receive 0 points. The ranking results from the sum of the points. Besides, the participants had the chance to leave comments as a freely written text.

We asked 12 stakeholders of the system to participate in the survey. These stakeholders were mainly internal stakeholders involved in the specification and development of the system. Eventually, 5 out of the 12 stakeholders we asked participated in the survey.

A. Results

The rankings of the individual participants are shown in Table I. The entries in the table are sorted in descending order of the participants preferences. If two representations are not separated by a horizontal line, the representations were ranked equally. Next to the name of the representation in the brackets are the decisions the participants made for the representation. The first number stands for the amount of times it was preferred, the second number for the amount of ties and the last number for the amount it was considered less useful.

The grouping representation is among the highest ranked as it appears three times at the top position. The normal form appears at the top position once and is ranked in the lower half by the rest of the participants. The original (currently used) representation is ranked first by one participant, second by another participant, and third by all other participants. The tree representation is once ranked first and twice on each the second and the third rank. Without exception, the participants all ranked the exact equivalent representation as the least preferable textual representation.

Table II shows the aggregated results for all participants. The first column shows the aggregated ranking by combining all the decisions taken by the participants. The number in the brackets next to the representation is the result of the computation mentioned for the individual participants applied to all decisions. The second column shows an aggregation based on an assigning of points for the individual ranks. A textual representation on the first rank receives five points, while the one on the last rank receives one. If two representations have the same rank, the points of the respective ranks are summed up and divided by the number of representations of the same rank. For example, for *Participant 1*, the representation on the third rank would receive three points and the representation on the fourth rank would receive two points. As there are two representations on the same rank both receive 2.5 points (the average). The resulting points are shown in the brackets next to the name of the representation. The resulting rankings for both ways of aggregating the decisions are the same. Nonetheless, the separations of the representations differ between the aggregations. While the ranking by points is close together, the representations in the combined ranking are farther apart. Especially, the grouping representation is far ahead in the combined aggregation, due to the fact, that it was preferred in most pairwise decisions.

Besides the pairwise comparisons, one participant used the opportunity to leave a comment. *Participant 2* mentioned that the grouping representation improves readability since confusing repetitions of elements are avoided.

B. Discussion

Based on the high ranking that the grouping representation achieved in both the individual and the aggregated rankings, we conclude that this is the most appropriate representation for the participants we asked. The high acceptance of this representation may have resulted from the fact that it resembles the structure of the currently used (original) representation and is thus familiar to the participants. On the other hand, it mitigates some of the weaknesses of the original representation such as non-atomic entries and unclear logic relations between the entries. Due to the similarity with the original representation, it is easy for participants to comprehend the grouping representation. As such, the grouping representation was most likely perceived as an improved version of the currently used representation.

The exact equivalent representation was ranked as the least preferable. We conclude, that a mere transformation of an activity into text is not suitable in a requirements document. This is most likely caused by the reduced readability, which makes the document harder to understand. This is also in accordance with the fact that graphical models are used to improve understandability in the first place.

Nevertheless, both the normal form and the tree representation, achieved good rankings for individual participants. As a consequence, as long as all representations are kept consistent with each other, the representations could exist side by side as views of the same function. This approach also has the advantage that implicit information in the models can be made explicit depending on the individual needs and the background of each stakeholder.

Based on the survey results, we clearly see that textual and graphical representations of models serve different purposes and thus, it is reasonable to perform research with the goal to assess how “good” textual representations of requirements models look like.

C. Threats to Validity

Our findings can only give a first impression on how an appropriate textual representation should look like.

In our survey, we considered only internal stakeholders within one company that are involved in the specification and development of one particular system. However, the textual representations must also be read and understood by stakeholders outside the company (e.g., legal authorities, contractors, customers), who may have different preferences with respect to textual representations. Also we received only a relatively small number of answers.

TABLE I: Ranking of textual representations for each participant in descending order

Participant 1	Participant 2	Participant 3	Participant 4	Participant 5
Normal Form (4:0:0)	Grouping (4:0:0)	Grouping (4:0:0)	Grouping (3:1:0)	Original (4:0:0)
Grouping (3:0:1)	Tree (3:0:1)	Original (3:0:1)	Tree (3:1:0)	Tree (3:0:1)
Original (1:1:2)	Original (1:1:2)	Tree (1:1:2)	Original (2:0:2)	Grouping (2:0:2)
Tree (1:1:2)	Normal Form (0:2:2)	Normal Form (1:0:3)	Normal Form (0:1:3)	Normal Form (1:0:3)
Exact Equivalent (0:0:4)	Exact Equivalent (0:1:3)	Exact Equivalent (0:1:3)	Exact Equivalent (0:1:3)	Exact Equivalent (0:0:4)

TABLE II: Aggregated ranking of textual representations for all participants

Rank	Combined	Points
1	Grouping (13)	Grouping (21.5)
2	Tree (5)	Tree (18)
3	Original (4)	Original (17.5)
4	Normal Form (-5)	Normal Form (12.5)
5	Exact Equivalent (-17)	Exact Equivalent (5.5)

We combined our survey with another study on quality issues that arise from using activity diagrams and their textual representations side by side [13]. Thus, all participants of this survey also participated in the previous study. Since some of the identified quality issues are linked to the currently used textual representation, the participants may be biased towards their opinion to the original textual representation. On the other hand, the participants are also familiar with the original representation form, which may also have an impact on its perception.

We did not consider the layout of the textual descriptions explicitly. There may be multiple layouts for the representations. For example, the grouping representation uses curly brackets, although they are not necessarily required to structure the document. Still, the representations are implemented in the requirements management tool that the participants use.

The purpose of the activity diagrams as used at Daimler is mainly for describing logical conditions that must hold in order to activate a function. The focus on propositional logic is based on that purpose. We do not know whether the choice of textual representation would be similar for activity diagrams that are used in other contexts. Thus, the generalizability of our findings is limited.

VI. Conclusion and Outlook

In this paper, we presented the textual representation of the sort of activity diagrams that is currently used by Daimler in requirements documents. Additionally, we presented four more textual representations as possible alternatives. We show on which aspects of an activity the different textual representations focus and their advantages and disadvantages. In a comparative survey, the textual representations were ranked by five stakeholders. Although the currently used textual representation reached high agreement, a majority of the participants prefer another textual representation. The representation that mitigates some of the weaknesses of the currently used representation yielded the best results, although it requires a more complicated document structure and thus might be harder to understand.

To validate the results, it is necessary to repeat the survey with more participants and in different contexts. It would also provide additional insight if we elicit qualitative feedback on why the respondents preferred one option over the other. A follow-up investigation on the reasons is planned. Besides, there are countless of imaginable representations in terms of the considered aspect and the used layout. As a result the presented representations only cover a fraction of the possibilities. There might be use cases in requirements engineering, in which the focus on other aspects is more important. Hence, more investigation is needed to develop a greater variety of textual representations.

References

[1] M. Broy, "Challenges in automotive software engineering," in *Proceedings of the 28th international conference on Software engineering (ICSE)*. New York, NY, USA: ACM, 2006.

[2] L. Apfelbaum and J. Doyle, "Model based testing," in *Software Quality Week Conference*, 1997.

[3] A. Vogelsang, S. Eder, G. Hackenberg, M. Junker, and S. Teufl, "Supporting concurrent development of requirements and architecture: A model-based approach," in *Proceedings of the 2nd International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, 2014.

[4] C. Reuter, "Variant Management as a Cross-Sectional Approach for a Continuous Systems Engineering Environment," in *Proceedings of the 8th Grazer Symposium Virtual Vehicle*, 2015.

[5] A. M. Davis, *Just Enough Requirements Management: Where Software Development Meets Marketing*. New York, NY, USA: Dorset House Publishing Co., Inc., 2005.

[6] E. Sikora, B. Tenbergen, and K. Pohl, "Industry needs and research directions in requirements engineering for embedded systems," *Requirements Engineering*, vol. 17, no. 1, 2012.

[7] N. A. Maiden, S. Manning, S. Jones, and J. Greenwood, "Generating requirements from systems models using patterns: a case study," *Requirements Engineering*, vol. 10, no. 4, 2005.

[8] J. Arlow, W. Emmerich, and J. Quinn, "Literate Modelling — Capturing Business Knowledge with the UML," in *International Conference on the Unified Modeling Language*. Springer, 1998.

[9] J. Arlow and I. Neustadt, *Enterprise patterns and MDA: Building better software with archetype patterns and UML*. Addison-Wesley Professional, 2004.

[10] R. F. Goldsmith, *Discovering Real Business Requirements for Software Project Success*. Artech House, 2004.

[11] Object Management Group (OMG), "OMG Unified Modeling Language (OMG UML), Version 2.5," OMG Document Number formal/2015-03-01 (http://www.omg.org/spec/UML/2.5/), 2015.

[12] D. Firesmith, "Generating Complete, Unambiguous, and Verifiable Requirements from Stories, Scenarios, and Use Cases." *Journal of Object Technology*, vol. 3, no. 10, 2004.

[13] M. Beckmann, A. Vogelsang, and C. Reuter, "A Case Study on a Specification Approach using Activity Diagrams in Requirements Documents," in *25th IEEE International Requirements Engineering Conference (RE)*, 2017.

[14] M. Beckmann, V. Michalke, A. Vogelsang, and A. Schlutter, "Removal of Redundant Elements within UML Activity Diagrams," in *ACM/IEEE 20th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2017.

[15] Object Management Group (OMG), "OMG Human-Usable Textual Notation (HUTN) Specification, Version 1.0," OMG Document Number formal/04-08-01 (http://www.omg.org/spec/HUTN/1.0), 2004.

[16] J. Nicolás and A. Toval, "On the generation of requirements specifications from software engineering models: A systematic literature review," *Information and Software Technology*, vol. 51, no. 9, 2009.

[17] T. Mens, G. Taentzer, and D. Müller, "Model-Driven Software Refactoring," *Model-Driven Software Development: Integrating Quality Assurance*, 2008.

[18] F. Meziane, N. Athanasakis, and S. Ananiadou, "Generating Natural Language specifications from UML class diagrams," *Requirements Engineering*, vol. 13, no. 1, 2008.

[19] C. Robinson-Mallett, "An Approach on Integrating Models and Textual Specifications," in *2nd International Model-Driven Requirements Engineering Workshop (MoDRE)*, 2012.

[20] B. Berenbach, "The Automated Extraction of Requirements from UML Models," in *11th IEEE International Requirements Engineering Conference (RE)*, 2003.

[21] D. Drusinsky, "From UML Activity Diagrams to Specification Requirements," in *IEEE International Conference on System of Systems Engineering (SoSE)*, 2008.

[22] D. Flater, P. Martin, and M. Crane, "Rendering UML activity diagrams as human-readable text." NISTIR 7469, National Institute of Standards and Technology, Tech. Rep., 2009.

[23] Object Management Group (OMG), "OMG Action Language for Foundational UML (Alf) Specification, Version 1.0.1," OMG Document Number formal/13-09-01 (http://www.omg.org/spec/ALF/1.0.1), 2013.

[24] R. Gilberg and B. Forouzan, *Data Structures: A pseudocode approach with C*. Nelson Education, 2004.