Semantic Integration of System Specifications to Support Different System Engineering Disciplines

Alexander Rauh, Wolfgang Golubski
Westsächsische Hochschule Zwickau
University of Applied Science
Dr.-Friedrichs-Ring 2A, Zwickau, Germany
{alexander.rauh,wolfgang.golubski}@fh-zwickau.de

Stefan Queins
SOPHIST GmbH
Vordere Cramergasse 13, Nuremberg, Germany
stefan.queins@sophist.de

***Abstract*—The assurance of high quality requirements and system specifications takes a huge effort and requires semantically integrated requirements. Especially when using different representations of requirements, the semantic integration is quite difficult. Today's requirements management tools integrate requirements only syntactically and, thereby, provide only limited features to support the requirements analyst during quality assurance. The following paper describes a concept for the semantic integration of requirements documented in different representations into one common model. This model allows analysing and measuring the quality of the integrated requirements by applying algorithms. The mentioned integration concept is applied to a use case driven requirements elicitation and analysis process using different representations like several UML diagrams or even template based textual requirements. The foundation to measure the quality of the integrated requirements is explained.**

Keywords—Specification Integration, Requirements Integration, Requirements Documentation, Requirements Modelling, Specification Quality, Requirements Quality

I. Introduction

In systems engineering the requirements of a system, as "a coherent, delimitable set of components that – by coordinated action – provides services" [1], are the foundation for other disciplines during the whole system's life cycle [2]. During development, these requirements are used for designing a system's architecture, for realizing a system's functionality during implementation and for defining a system's quality aspects. Furthermore, quality assurance uses requirements as foundation to define test cases which should identify mistakes in the implementation or which are used to verify a system's functionality. Some concepts even try to transform the requirements documentation directly into source code [3] or simulate the system under consideration before development [4]. These concepts assume high quality requirements documentation but do not provide mechanisms to ensure this required quality.

In system engineering projects the components of a system are usually divided into software, electronic, mechanic and optic parts etc. depending on the system under consideration. For the documentation of the requirements of these parts usually different representations are used. For example, software development often uses use case centric and process oriented documentation types like UML use case diagrams in combination with activity diagrams. For the behaviour of the electronic components state oriented representations like UML state charts or more formal notations like MATLAB Simulink and temporal logics are used. Dividing the system into several views is an already known approach for software systems [5]. One of the biggest challenges of these views is to ensure the quality of each view as well as the overall quality of the specification.

Requirements management tools like PTC Integrity, IBM Rational DOORS or Sparx Enterprise Architect allow using different notations for requirements documentation. But these tools only integrate the mentioned notations syntactically into one common database. The information described by these several notations has different sources because there is no common database for the semantically integrated requirements. For example, a textual requirement is stored as one object and a use case of the system under consideration is stored as another object. Assumed that both objects address the same service of the system there are two sources of the same information. The change of one source leads to a contradiction into the system specification. Ideally there is only one source which is used in several notations.

The approach discussed in this paper provides a concept to integrate several representations of requirements semantically in order to measure the quality of a system specification according to the definition of requirements characteristics of IEEE 29148:2011 [6].

In the first section after this introduction some related works are discussed. These related works also integrate requirements or aim at the identification of defects within system specifications. In the third section the idea of the semantic integration of requirements is explained. The fourth and the fifth section describe the details of the semantic requirements integration for a use case driven requirements analysis process. Documentation rules and assumptions as foundations and one meta-model for the semantic integration of UML activity diagrams are explained. The sixth section shows parts of a tool implementation of the presented approach in order to provide a proof of concept. An example will show the idea of the semantic integration of several subsystems into one common specification. After that, the benefits are listed followed by a discussion of topics for further researches.

0-7695-6394-5/18/$31.00 ©2018 IEEE
DOI 10.1109/SOSE.2018.00016

II. Related Work

There are different approaches which aim to ensure the quality of requirements like consistency and completeness. Some of these approaches define specific meta-models for requirements or even transform the requirements documentation directly into source code. No approach integrates several representations of requirements semantically in order to measure the quality characteristics. Some approaches integrate the representations only syntactically.

The Unified Requirements Modeling Language (URML) is a dedicated language for requirements modeling which combines concepts from hazard analysis, product line and goal modelling [7]. The URML is defined by a separate meta-model and does not integrate more common meta-models for requirements modelling like the UML and its diagrams [8]. The advantages and concepts for requirements modelling of already known notations cannot be used. Furthermore, there are no possibilities to measure the quality of the requirements models.

The second approach uses relations between the requirements of a specification to identify inconsistencies and incompleteness [9]. These traces add some semantic to the syntactically integrated requirements. The requirements engineer has to manage the traces and relations including different relation types between the single requirements manually. Hence, there is no formal method to support the generation of content-related and correct traces. The meta-model in [9] is similar to the meta-model of the Requirements Interchange Format (ReqIF) [10]. Both meta-models provide some artefacts for requirements management such as relations and attributes for the requirements. But meta-models do not integrate the content described by these requirements.

Another approach applies graph transformation algorithms using UML use case diagrams, activity diagrams and collaboration diagrams as a foundation [11]. Collaboration diagrams contain a domain model to define the pre- and the post-conditions of each use case and each step of this use case separately. The use cases and the actions transform the preconditions into the related postcondition. The mentioned graph transformation uses the preconditions, applies the formal transformation rules derived from the use cases or actions and matches the result of this transformation to the postcondition of the rule related action. Mismatches between the transformation result and the defined postcondition represent the inconsistencies within requirements. This approach is limited to the three listed UML diagram types. There are no possibilities to use further notations.

Other approaches consider less common representation types of requirements to check the quality, especially the consistency, of a system specification. Moreover, there are no possibilities to add further notations as foundation for the analysis purposes.

Another approach applies algorithms for consistency checks to the formal Software Cost Reduction (SCR) tabular notation, but does not support more common requirements representation types like UML diagrams or textual requirements [12]. Thereby, before applying this approach onto a specification, this specification has to be transformed into the SCR notation which causes additional effort.

The transformation of textual requirements into Essential Use Cases (EUC) is described in [13]. These captured EUCs are matched to existing EUC pattern describing abstract system interactions. The identified mismatches are inconsistencies of the former transformed textual requirements. Only high level textual requirements are supported by this approach. Furthermore, the manual transformation of textual requirements into EUC may lead to mistakes. The concept in [13] is limited to high level textual requirements. Other representation types like UML are not supported as foundation.

The last related approach analyzes the consistency of requirements by capturing textual requirements, creating UML models from these requirements manually and converting these models into a problem ontology [14]. This problem ontology is analyzed for consistency via reasoning and is matched to a domain ontology representing the domain knowledge related to the system's domain to discover contradictions. UML use case diagrams support only abstract views onto a system. For detailed views onto the system's functionality other UML diagram types like activity diagrams or state charts have to be used, but are not supported [14].

Approaches which use formal representation types of requirements like VHDL, MATLAB Simulink or different temporal logics for quality analysis are not discussed. At the moment, the realization of the integration concept focuses Use Case driven requirements analysis processes using less formal notations to show its benefits. If there are some significant results of the requirements integration, more formal notations like VHDL or MATLAB Simulink will be added to the concept.

III. Semantic Integration of Requirements

The idea of this approach is to integrate a set of requirements specified in several representations semantically into a common database. This database implements a meta-model which defines the structure and the semantic of the relevant information for specifying a system. Algorithms analyse the content of this database and evaluate the quality of the integrated requirements. Furthermore, this approach allows integrating specifications of a system and its subsystems semantically in order to check the completeness, consistency etc. of these specifications. Fig. 1 visualizes this concept. The representation types on top of this figure are only some examples to explain the idea:

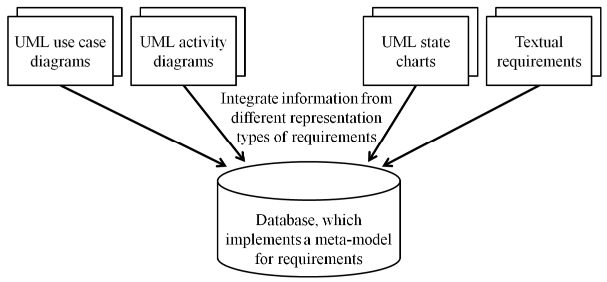
UML use case diagrams
UML activity diagrams
UML state charts
Textual requirements
Integrate information from different representation types of requirements
Database, which implements a meta-model for requirements

Fig. 1. Integration of the content of requirements documentation.

As shown in Fig. 1, there may be many different representation types for requirements to be integrated depending on the specific development project. In order to be independent of representation type specific constraints and due to the expandability by other representation types, the integration concept mentioned above is divided into the three layers shown in Fig. 2.

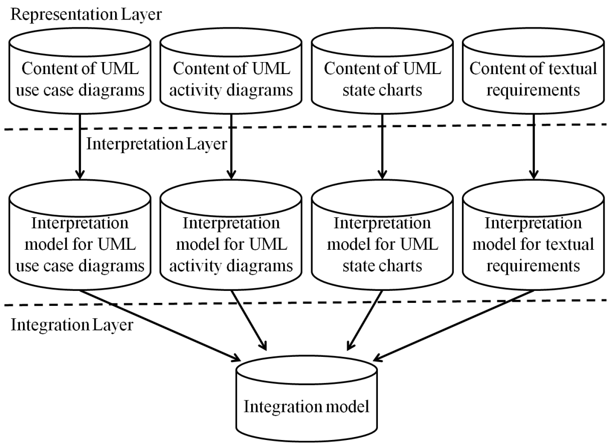
Representation Layer
Content of UML use case diagrams
Content of UML activity diagrams
Content of UML state charts
Content of textual requirements
Interpretation Layer
Interpretation model for UML use case diagrams
Interpretation model for UML activity diagrams
Interpretation model for UML state charts
Interpretation model for textual requirements
Integration Layer
Integration model

Fig. 2. Layer concept of requirements integration with Representation Layer, Interpretation Layer and Integration Layer.

The Representation Layer contains the requirements documentation consisting of different notation. The content implements different meta-models e.g. the UML meta-model for activity diagrams, use case diagrams, class diagrams and state charts defined by UML [15] or even textual requirements. The meta-models within the Representation Layer are predefined by the specification language and remain unchanged for integration purposes.

In order to measure the quality of the specification there are rules and assumptions to use the notations within the Representation Layer. These rules and assumptions depend on the requirements analysis process. Violations decrease the quality of the specification. Some of these rules and assumption are explained in the fourth section of this article.

The second layer is the Interpretation Layer which contains interpretation models related to the representation types of the requirements documentation. These interpretation models encapsulate representation type specific content from the information to be integrated. Especially, critical violations of documentation rules which prevent the requirements integration are encapsulated in this layer. Furthermore, the interpretation layer allows measuring the quality of each representation independently. There is one interpretation meta-model for each representation which could be used for integration purposes.

If this approach should be applied to other representation types or if other interpretations are necessary, new interpretation meta-models can be defined or the existing interpretation meta-models can be adopted.

The third layer is the Integration Layer. This layer contains the integrated function-oriented meta-model for requirements which describes the structure of the integrated information to specify a system. This function-oriented meta-model for requirements is explained in detail in another paper [16]. A database, which implements this integration meta-model, stores the semantically integrated requirements. Algorithms analyse this database and measure the overall quality of the integrated specification.

In order to apply this concept to a system specification, a defined process is necessary. Before the requirements integration, the requirements analyst describes the requirements of a system conventionally e.g. by performing a use case analysis as defined in [17]. During analysis the analyst creates different instances of the supported representation types, possibly more than one instance of the same representation type. After the requirements analyst has finished the use case analysis the process shown in Fig. 3 can be applied. The actor, who performs the related action, is defined by the stereotype in the brackets.

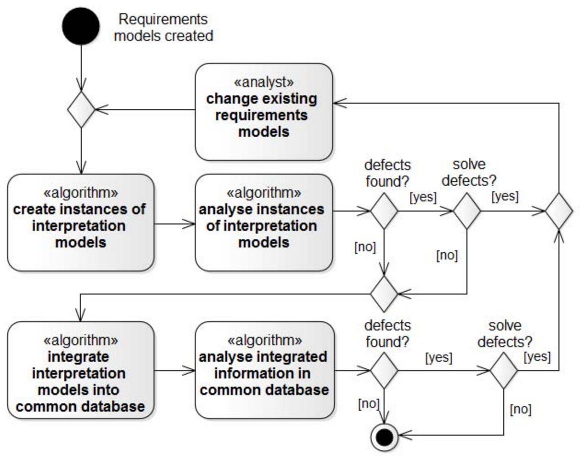
Requirements models created
«analyst» change existing requirements models
«algorithm» create instances of interpretation models
«algorithm» analyse instances of interpretation models
defects found?
[yes]
[no]
solve defects?
[yes]
[no]
«algorithm» integrate interpretation models into common database
«algorithm» analyse integrated information in common database
defects found?
[yes]
[no]
solve defects?
[yes]
[no]

Fig. 3. Process for the requirements integration.

In the first step *create instances of interpretation models*, algorithms generate instances of the representation type related interpretation models for each instantiated representation type. After this creation, algorithms *analyse instances of interpretation models*. During this step, violations of the representation related requirements documentation rules are checked and reported as defects. For example, if there is a use case without any related actor or an action without a name, a rule violation will be reported as a defect to the requirements analyst.

After this step and if the algorithms found any defects in the interpretation models, the requirements analyst has the possibility to solve these defects. Therefore, he changes existing instances of representation types and repeats the first two steps of the integration process.

If there are no further defects in the interpretation models or if the requirements analyst does not want to solve the identified defects, algorithms *integrate instanced interpretation models* *into common database*. Semantic interrelations between the information of the different notations will be added.

In the last step, algorithms *analyse integrated information in common database* for further violations of the documentation rules in the representation layer. Especially, the rules which aim at interrelations between different representation types are analysed. The results of this analysis are also reported to the requirements analyst. He can fix the overall defects or can add further instances of representation types. After that, the requirements integration could simply be repeated.

IV. Rules and Assumptions for Requirements Documentation

The concept described in the previous section is generic and could be applied to any requirements analysis process. For the application of this concept onto a specification the generic parts like the supported representation types for requirements, assumptions and rules for requirements documentation and the interpretation meta-models including the transformation rules have to be defined. This section explains these parts of the mentioned concept for a use case driven requirements analysis process as defined by [17] by example. Therefore, the following representation types are supported:

- UML use case diagrams
- UML class diagrams for information modelling
- UML activity diagrams
- UML state charts
- Textual functional and non-functional requirements
- Glossary entries

The use cases describe functions of the system under consideration at the highest level of abstraction and serve as origin for other artefacts of the specification. Activity diagrams refine these use cases from a process oriented point of view. State charts also refine use cases but focus on the states of domain objects and the system under consideration as a whole. Textual functional and non-functional requirements specify other artefacts more precisely and add further information. Class diagrams and glossary entries will be used to define the common language of the specification. Furthermore, class diagrams specify the objects of the system's domain with their attributes and relations. A specification could consist of many artefacts of several representation types. Furthermore, more than one artefact of the same representation type is possible.

After the definition of the supported notations for requirements integration the assumptions and rules for using these notations have to be defined.

Assumptions shall prevent issues that cannot be checked by algorithms but which have to be fulfilled for the requirements integration or for measurement of the specification quality. The requirements analyst has to ensure these assumptions without any tool support. One example for such an assumption is: Each use case in the requirements model has to be of significant value for at least one actor of the system under consideration. This assumption is derived from the definition of use cases of Jacobsen [18]. If a use case does not provide any of such a value to an actor of the system, this use case and its specific refinements should not be part of the specification. In this case, the quality criteria necessity according to [6] is violated.

In contrast to the assumptions there are documentation rules which will be checked by algorithms during the requirements integration or the analysis of requirements quality. This approach separates between representation specific and representation comprehensive documentation rules. Representation specific rules address the usage of artefacts of one representation type and are checked independent of other artefacts. Thereby, each artefact is evaluated separately and defects are encapsulated of the further integration process. The representation comprehensive rules address the semantic interrelations between the different notations and its model elements. After the integration of the interpretation models into the common database algorithms evaluate the integration model for violations against these comprehensive documentation rules. For the evaluation of the overall quality of the integrated specifications the representation specific defects and the comprehensive defects are aggregated.

In order to give an idea of the structure and content of the documentation rules three different rules are explained. Each rule consists of a rule text, at least one addressed model element, one or more assigned quality characteristics and a criticality. If a rule is violated, the assigned quality characteristic for the related type of model elements will be decreased. The criticality describes the impact of a rule violation onto the further integration process. It is separated into low and high criticality. Low critical violations do not prevent the further integration process for the related model element. A high critical violation prevents the integration of the related model element into the common database.

Rule 1: Names of UML actions and activities follow the structure <verb> [adjective] <noun> [adverb].

Addressed model elements:	UML activities, actions
Assigned quality characteristics:	syntactic correctness
Criticality:	high

Rule 1 shows such a representation specific rule which addresses the naming of activities and actions in UML activity diagrams. In order to define unambiguous process steps the names of activities and actions should consist of a verb, a noun and optionally an adverb and an adjective. The verb specifies a process realized by the system under consideration. The noun represents a domain object or a domain object attribute that is involved in the process. The adjective will be interpreted as a state of this domain object. The adverb specifies additional information of the process like quality of service.

In a first realization there will be no grammar or part of speech checks. Furthermore, if a domain object consists of more than one token this domain object has to be named in camel case as one token. For example, the domain object "contact list" of a smartphone has to be named like

"ContactList". After a proof of concept the realization can be extended by natural language processing (NLP) tools like [19] to add part of speech and grammar checks. Thereby, the assumptions for the interpretation of names will be avoided.

The assigned quality characteristic of rule 1 is the syntactic correctness. If an activity or an action violates this rule, the correctness of the activity diagrams and the correctness of the whole specification are decreased. Additionally, such a syntactic defect prevents the further integration process for the related model element. Thereby, the criticality of rule 1 is high.

Rule 2: The noun in the name of an activity or an action has to be defined as a class, an attribute or role name within the Information Model.

Addressed model elements:	UML activities, actions, classes
Assigned quality characteristics:	completeness, unambiguity
Criticality:	low

Rule 2 shows a representation comprehensive documentation rule which addresses the interrelation between UML activities and actions with the information model of the system under consideration. In order to provide an unambiguous specification each domain object has to be defined as a class within the information model. The missing class definition decreases the completeness and the unambiguity of the specification. A violation of this rule has no influence onto the requirements integration process, so its criticality is set to *low*.

Rule 3: Each activity refines one use case or is used as a call-behaviour within another activity.

Addressed model elements:	UML activities, use cases
Assigned quality characteristics:	consistency, completeness, necessity
Criticality:	low

The representation comprehensive documentation rule 3 defines the usage of UML activities to refine use cases or to specify a call-behaviour action. If there is an activity which does not refine any use case or which is not used as a call-behaviour, there is a contradiction within the specification. The specification may be incomplete or the violation related activity may be not necessary. It depends on the decision of the requirements analyst. Criticality is also low because the integration of the related activity is not prevented.

Rule 4: If a use case A extends an use case B, the activity of use case A has to be part of the activity of use case B as a call-behaviour action.

Addressed model elements:	UML activities, use cases
Assigned quality characteristics:	consistency, completeness, unambiguity
Criticality:	low

Rule 4 also describes a representation comprehensive documentation rule. This rule was derived from the semantic of extend relations between two use case. If there is an extend relation between two use cases, there is also a relation between their related activities. The activity of the extending use case has to be called of one action within the activity of the extended use case. If there is no such call-behaviour, the completeness of the specification is decreased. Additionally this incompleteness leads to an inconsistency between the use case view and the control flow oriented view onto the system under consideration. The specification is ambiguous.

For the proof of concept for integrating a use case driven specification according to [17] there are 27 representation specific and 13 representation comprehensive documentation rules defined yet.

V. Interpretation Meta-Models and Integration Meta-Model

The idea of the interpretation meta-models is to prepare the representation models for the integration. Additionally, the quality of these representation models can be measured separately. The requirements analysis process and the documentation rules in the previous section influence the interpretation meta-models. If another analysis process is used or if there are other project specific documentation rules the interpretation meta-models may be adjusted. For the realization of the requirements integration using the use case driven requirements analysis process defined in [17] interpretation meta-models for the representation types listed in the previous section are defined.

In order to give an idea of the structure of these interpretation meta-models the interpretation meta-model for UML activity diagrams is explained in detail. Fig. 4 shows the interpretation for executable nodes of UML activity diagrams, e.g. the structure of the names of actions. The interpretation of control nodes and object nodes complete this meta-model, but are not described within this paper.

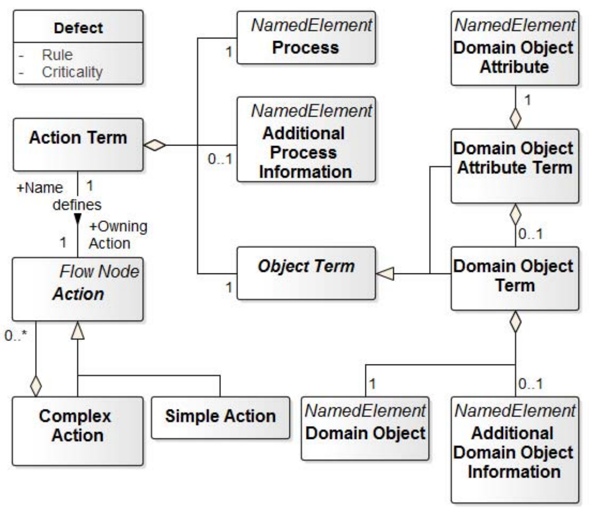

Fig. 4. Interpretation meta-model for executable nodes of UML activity diagrams.

The abstract *Action* class specializes a *Flow Node* and represents the most significant element of this meta-model. An action is either a *Complex Action* or a *Simple Action*. Such a simple action defines an atomic behavior of the system under consideration and cannot be refined by other actions. In contrast, a complex action specifies behavior of the system under consideration, which can be refined by further actions. Call behavior as well as activities of UML activity diagrams are interpreted as complex actions. Both kinds of actions are defined by one *Action Term* representing the name of the action within a UML activity diagram. In contrast to the UML and similar to the use case interpretation meta-model the content of actions names is specified in more detail. An action term consists of one *Process* defined by a verb, one *Object Term* and optionally one *Additional Process Information*. The additional process information adds some details to the related process like quality of service.

The activity diagram interpretation meta-model separates between a *Domain Object Term* and a *Domain Object Attribute Term* as the process related *Object Term*. A domain object term consists of one *Domain Object* and optionally *Additional Domain Object Information* which will be interpreted as a state of the related domain object. A domain object attribute term consists of one *Domain Object Attribute* and optionally of a domain object term in order to reference the owning domain object. The structure of the *Action Term* is the result of rule 1 of the previous section.

The structure and the semantic of the integration meta-model implemented by the common database is described in [16] in detail. But in order to explain the integration of UML activity diagrams, the relevant elements of the integration meta-model are shown in Fig. 5. Additionally, the transformations between the interpretation model of activity diagrams and the common database are explained.

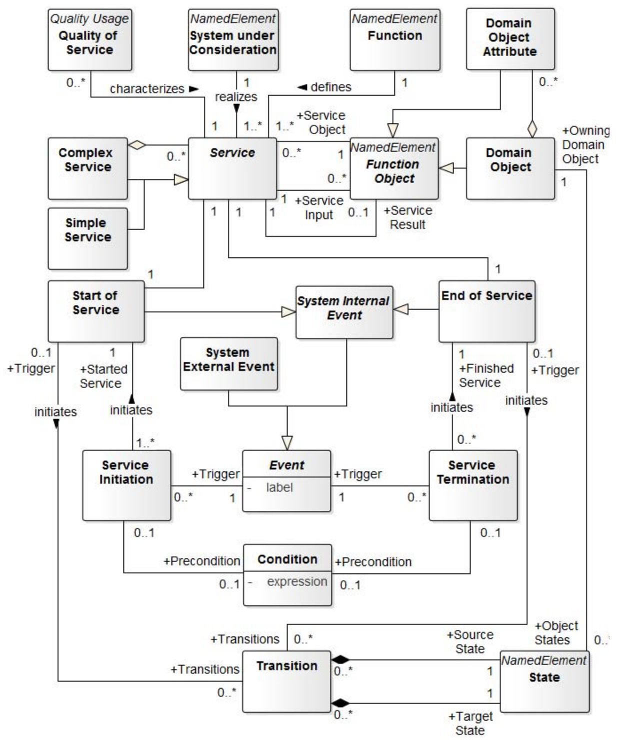

Fig. 5. Services and states of domain objects within the Integration Meta-Model

Simple Actions of the activity diagram interpretation model will be transformed to *Simple Services* within the integration model. *Complex Actions* are integrated to *Complex Services*. Use cases, for example, are also integrated to such *Complex Services*. The *Process* of an *Action* is transformed to the *Function* of a *Service*. Depending on the structure of the *Object Term* within the activity diagram interpretation model a *Domain Object* or a *Domain Object Attribute* within the integration model will be generated. *Domain Objects* and *Domain Object Attributes*, for example, are derived from classes and properties in class diagrams too. *Additional Process Information* is integrated to *Quality of Service*.

The interpretation model elements derived from control nodes of activity diagrams are transformed to *Start* and *End of Services*, *Service Initiation* and *Service Termination* including the related *Condition*. As described in [16], influence the services of a system under consideration the states of domain objects of this system. Thereby, the semantic integration of use case diagrams, activity diagrams and state charts is realized.

VI. Example for the Semantic Integration

This section explains the idea of the semantic integration of several specifications of one system and its subsystems into one common system specification in order to check the completeness and consistency. A comfort relaxation function of a today's premium vehicle serves as an example. In our example, this function massages a passenger, adjusts the ambient light of the vehicle and plays dedicated relaxation music. The *Comfort Relaxation* system uses the subsystems

Player, *Seats* and *Ambient Light*. In this example, the system and the three subsystems are specified in several specifications. The specifications of *Comfort Relaxation*, *Player* and *Seats* follow a process oriented specification paradigm using UML use cases and activity diagrams. The *Ambient Light* subsystem is specified by a state oriented paradigm. The foundations for the semantic integration are the use cases of the Comfort Relaxation System shown in Fig. 6 and the use cases of the subsystems listed in Fig. 7. The integration of the use case *Start ComfortRelaxation* of Fig. 6 is explained in this section. Therefore, this use case has associations to the three subsystems as actors.

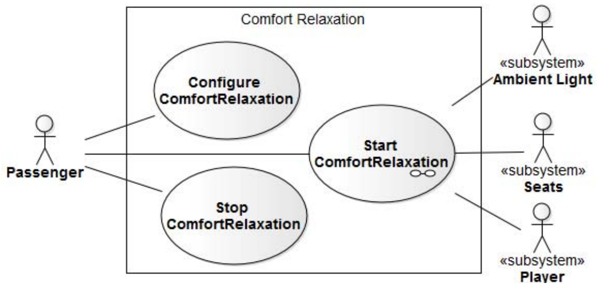

Fig. 6. Use Cases of the Comfort Relaxation system.

As an additional basis, the associations of the subsystem use cases *Playback Title*, *Massage Passenger* and *Adjust Illumination* to the Comfort Relaxation system as actor of these use cases shown in Fig. 7 are used.

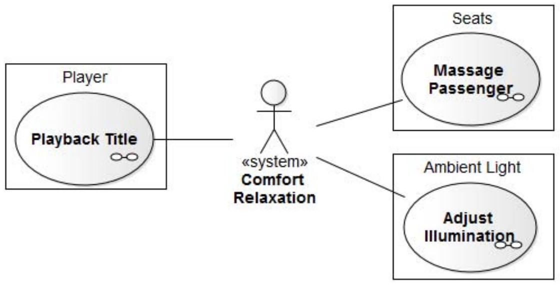

Fig. 7. Use Cases of the three subsystems Player, Seats and Ambient Light.

The simple control flow of the activity diagram shown in Fig. 8 is used as the refinement of the *Start ComfortRelaxation* use case. The actions within this flow are used as further foundations for the semantic integration.

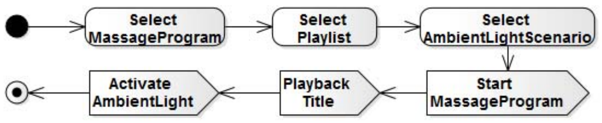

Fig. 8. Control flow of the Start ComfortRelaxation use case.

The actions *Select MassageProgram*, *Select Playlist* and *Select AmbientLightScenario* describe functions to configure the comfort relaxation function. These actions could be refined by further activity diagrams. But in order to explain the integration process, these refinements are unnecessary and are not part of this example section. The use cases *Playback Title*, *Massage Passenger* and *Adjust Illumination* of the three subsystems are more relevant for the semantic integration. The process shown in Fig. 9 could be a refinement of the *Massage Passenger* use case of the *Seats* subsystem.

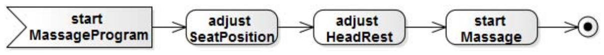

Fig. 9. Process of the Massage Passenger use case of the Seats subsystem.

The use case *Playback Title* of the *Player* subsystem could be refined by the activity diagram shown in Fig. 10.

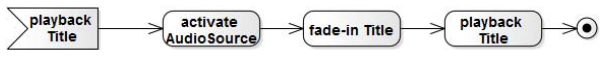

Fig. 10. Process of the Playback Title use case of the Player subsystem.

In contrast to the process oriented specifications of *Comfort Relaxation*, *Seats* and *Player* is the specification of the *Ambient Light* subsystem state oriented. Fig. 11 shows the states and transitions of *Ambient Light* for the mentioned example. This state chart follows a modelling pattern. The use cases of Ambient Light are modelled as do-behaviour elements within the several system states. The guards check the executed use cases for their end event and lead to state transitions.

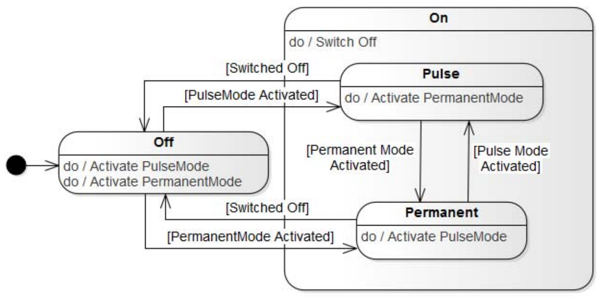

Fig. 11. States and transitions of the Ambient Light subsystem.

The system and its subsystems are specified using UML diagrams. Thereby, the integration of this short example seems to be quite easy and could be realized using a simple import function of a common UML modelling tool. But such an import function integrates the several parts only syntactically into one common UML model. The challenge is to integrate the information of the several models semantically into one common model in order to provide compatibility between the four specifications of the example. This semantic integration would support the requirements analysts of the several systems during the definition of the interfaces.

The most relevant results of the semantic integration are shown in Table 1. This table shows the integrated services of the Comfort Relaxation system and the three subsystems.

TABLE I. SYSTEM SERVICES AS RESULTS OF THE SEMANTIC INTEGRATION

ID	Service		Subject	Triggered by Service	Triggers Service
	Process	Domain Object			
S1	start	ComfortRelaxation	Comfort Relaxation	-	-
S2	select	MassageProgram	Comfort Relaxation	S1	S3
S3	select	Playlist	Comfort Relaxation	S2	S4
S4	select	AmbientLightScenario	Comfort Relaxation	S3	S5
S5	start	MassageProgram	Comfort Relaxation	S4	S11
S6	playback	Title	Comfort Relaxation	S5	S14
S7	**activate**	**AmbientLight**	**Comfort Relaxation**	**S6**	**missing**
S8	stop	ComfortRelaxation	Comfort Relaxation	-	-
S9	configure	ComfortRelaxation	Comfort Relaxation	-	-
S10	massage	Passenger	Seats	-	-
S11	start	MassageProgram	Seats	S5	S12
S12	adjust	SeatPosition	Seats	S11	S13
S13	adjust	HeadRest	Seats	S12	S14
S14	start	Massage	Seats	S14	-
S15	playback	Title	Player	-	-
S16	playback	Title	Player	S6	S17
S17	activate	AudioSource	Player	S16	S18
S18	fade-in	Title	Player	S17	S19
S19	playback	Title	Player	S18	-
S20	adjust	Illumination	Ambient Light	-	-
S21	**activate**	**PulseMode**	**Ambient Light**	**-**	**-**
S22	**activate**	**PermanentMode**	**Ambient Light**	**-**	**-**
S23	**switch-off**	**AmbientLight**	**Ambient Light**	**-**	**-**

Use cases are integrated as complex services into the integration model due to the high level of abstraction of use cases. Thereby, the services S1, S8, S9, S10, S15 and S20 are complex services. The other services are simple services because they are derived from atomic actions of the three activity diagrams or were defined as do-behaviour within the state chart of the Ambient Light subsystem.

The first defect of the four specifications is the missing value in the *Triggers Service* column of Table 1. The service *S7* is the result of the integration of the signal send of the control flow of the *Start ComfortRelaxation* use case shown in Fig. 8. The integration model expects at least one triggered and integrated service. There is no triggered service integrated because there is no corresponding signal received within any of the activity diagrams nor a do-behaviour within any state chart of the mentioned example. Corresponding means that the elements have the same name. This mentioned defect is the result of an inconsistency of the Complex Relaxation system specification and the specification of the Ambient Light subsystem.

Further defects are the missing triggers of the integrated services S21 to S23. As mentioned at the beginning of this section, the comfort relaxation function should include a massage, playback of music and an ambient light illumination. But there is no service integrated which will trigger the activation of the *PulseMode* or the *PermanentMode* of the *Ambient Light* subsystem.

Another benefit of this approach is the semantic integration of use cases, actions and states of domain objects. The integration would also identify defects within the interrelations between process oriented and state oriented views of the integrated specifications. Table 2 shows the integration results which focus the states of this short example. The integrated services S21 to S23 of Table 1 serve as triggers for the transitions. The defects of these three services were already explained.

TABLE II. INTERRELATIONS BETWEEN SYSTEM STATES AND SYSTEM SERVICES

ID	Domain Object	State	Transition Trigger	Condition	Target State
T1	Ambient Light	Off	S21	-	Pulse
T2	Ambient Light	Off	S22	-	Permanent
T3	Ambient Light	Pulse	S22	-	Permanent
T4	Ambient Light	Pulse	S23	-	Off
T5	Ambient Light	Permanent	S21	-	Pulse
T6	Ambient Light	Permanent	S23	-	Off

Due to the complexity and the scope of the semantic integration the mentioned example is very short and only some of the possibilities are described. For example, the semantic integration of the information models of the four specifications provides the possibility to get a comprehensive information model for the Comfort Relaxation system all its subsystems. This model may serve as a common language for the requirements analysts and, thereby, would support the definition of further interfaces between the systems.

VII. BENEFITS

With this approach, it is possible to integrate requirements semantically into a common database. The integration concept supports common representations for requirements like UML diagrams and even template based textual requirements. The semantically integrated information supports the requirements analyst to identify defects within his system specification automatically according to predefined documentation rules. Especially, in case a larger specification should be analysed, the automatic quality evaluation saves a lot of effort for the requirements analyst. Furthermore, the tool guarantees to identify all violated documentation rules. The mentioned concept supports quality measurements for one dedicated requirements representation within the system specification as well as the system specification as a whole. Especially, the checks of the semantic interrelations between the several representations provide a high benefit to the requirements analyst.

The adoptability of this approach to several analysis processes is guaranteed by the concept of the three layers mentioned in the third section. Depending on the specific analysis process and the project constraints the interpretation meta-models, the documentation rules as well as the transformation rules have to be customized. Theoretically with the investment of some effort, it is possible to use any representation type of requirements for semantic integration and quality analysis purposes.

Regarding the identified defects of a specification, the requirements analyst can decide which of these defects he wants to fix. Thereby, the level of the quality of a specification can be adjusted to the projects needs and constraints. For example, the level of quality for a specification in safety critical systems (e.g. cars or airplanes) has to be quite higher as in less safety critical systems (e.g. business software).

The process to use this integration concept is flexible and lightweight. It is applicable to common system development processes and supports more traditional as well as iterative and incremental processes with short iterations of only a few weeks. The integration of a system specification can be repeated easily with less effort. The requirements analyst has the possibility to improve the specification quality step by step.

VIII. Conclusion and Further Researches

This paper explains a concept for the semantic integration of information described by different representations of requirements. This concept is applied to a use case driven requirements elicitation and analysis process using UML, template-based textual requirements and glossary entries. Therefore, the following interpretation meta-models listed in the fourth section were defined. Additionally, the integration meta-model for the common database is defined and published.

The quality checks of a specification are realized in two dedicated steps in the requirements integration process. In the first step during instantiation of the interpretation models the transformation rules check the representation specific documentation rules. If such a rule was violated, the transformation rule generates a defect. After the instantiation of the interpretation models the requirements analyst gets a report of these defects and may fix them. In the second step for the quality checks algorithms analyse the integrated information in the integration model and generate comprehensive defects. The defects of both steps provide the foundation for a formal analysis of system specifications by metrics.

The concept described in this paper is implemented using a combination of Java, Eclipse EMF and ATL. Java, Eclipse EMF was used to implement the several meta-models in the Interpretation and the Integration Layer. ATL was used for the implementation of the model-to-model transformations between the three layers. Java provides the infrastructure to vary the order of the transformations steps. Additionally, the analysis of the defects and the measurements of the quality are realized in Java.

At the moment the validation of this concept is still in progress. The integration will be applied to several system specifications of the industry in order show its result and benefits. This validation will also provide issues for further improvements of the concept. The results of this validation will be presented later during research in a separate paper.

Some topics for further researches are already known. First of all, the definition of metrics to generate statistics for the quality of a system specification might be useful. These metrics would support the validation of this concept when comparing the integration results of a specification. Additionally, the statistics can be used to measure the maturity of a system specification during a requirements analysis process.

Another interesting question is the definition of views onto the integrated information of the integration model. Once integrated, the information can be used in reverse transformations to generate new representation models. Thereby, this approach would support the requirements analyst during analysis by generating new prefilled instances of supported representations.

The integration of the concept mentioned in the previous sections into a system development process like [4] is another topic for further researches. Especially, the transition of the integrated information to the architecture, design and testing disciplines during system development seems to be interesting. The improvement of the requirements quality when applying the concept above may also improve the quality of these other disciplines.

At the moment, the developed integration concept does not include formal notations like VHDL, MATLAB Simulink or temporal logics for requirements documentation. The support of these notations might be interesting to cover the development of technical systems. Thereby, the integration concept for system specifications would be more flexible and would address additional types of development projects.

References

[1] M. Glinz, *A Glossary of Requirements Engineering Terminology,* 1st ed. Available: https://www.ireb.org/content/downloads/1-cpre-glossary/ireb_cpre_glossary_16_en.pdf. Accessed on: Jun. 01 2016.

[2] D. D. Walden, G. J. Roedler, K. Forsberg, R. D. Hamelin, and T. M. Shortell, *Systems engineering handbook: A guide for system life cycle processes and activities,* 4th ed., 2015.

[3] M. Śmiałek and W. Nowakowski, *From Requirements to Java in a Snap: Model-Driven Requirements Engineering in Practice*. Cham: Springer International Publishing, 2015.

[4] K. Pohl, R. Achatz, H. Hönninger, and M. Broy, *Model-based engineering of embedded systems: The SPES 2020 methodology*. Berlin and New York: Springer, 2012.

[5] P. Kruchten, *The 4+1 View Model of Architecture*: IEEE Software.

[6] *Systems and software engineering -- Life cycle processes --Requirements engineering*, 2011.

[7] J. Helming *et al.,* "Towards a unified Requirements Modeling Language," in *2010 Fifth International Workshop on Requirements Engineering Visualization (REV)*, Sydney, Australia, pp. 53–57.

[8] F. Schneider, B. Bruegge, and B. Berenbach, "The unified requirements modeling language: Shifting the focus to early requirements elicitation," in *2013 International Comparing *Requirements* Modeling Approaches Workshop (CMA@RE)*, Rio de Janeiro, Brazil, pp. 31–36.

[9] A. Goknil, I. Kurtev, K. van den Berg, and J.-W. Veldhuis, "Semantics of trace relations in requirements models for consistency checking and inferencing," *Softw Syst Model*, vol. 10, no. 1, pp. 31–54, 2011.

[10] Object Management Group, Inc., *Requirements Interchange Format™ (ReqIF™),* 1st ed. Available: http://www.omg.org/spec/ReqIF/1.2. Accessed on: Feb. 13 2017.

[11] J. H. Hausmann, R. Heckel, and G. Taentzer, "Detection of conflicting functional requirements in a use case-driven approach: a static analysis technique based on graph transformation," in *the 24th international conference*, Orlando, Florida, 2002, p. 105.

[12] C. L. Heitmeyer, R. D. Jeffords, and B. G. Labaw, "Automated consistency checking of requirements specifications," *ACM Trans. Softw. Eng. Methodol.*, vol. 5, no. 3, pp. 231–261, 1996.

[13] M. Kamalrudin, J. Hosking, and J. Grundy, "Improving requirements quality using essential use case interaction patterns," in *Proceeding of the 33rd international conference*, Waikiki, Honolulu, HI, USA, 2011, p. 531.

[14] P. Kroha, R. Janetzko, and J. E. Labra, "Ontologies in Checking for Inconsistency of Requirements Specification," in *Third International Conference on Advances in Semantic Processing (SEMAPRO)*, Sliema, Malta, 2009, pp. 32–37.

[15] Object Management Group, Inc., *Unified Modeling Language,* 2nd ed. Available: http://www.omg.org/spec/UML/. Accessed on: Feb. 10 2017.

[16] A. Rauh, W. Golubski, and S. Queins, "A requirements meta-model to integrate information for the definition of system services," in *2017 IEEE Symposium on Service-Oriented System Engineering*: IEEE / Institute of Electrical and Electronics Engineers Incorporated, 2017.

[17] T. Cziharz, P. Hruschka, S. Queins, and T. Weyer, *Handbook of Requirements Modeling IREB Standard,* 1st ed. Available: https://www.ireb.org/content/downloads/17-handbook-cpre-advanced-

level-requirements-modeling/ireb_cpre_handbook_requirements-modeling_advanced-level-v1.1.pdf. Accessed on: Jun. 01 2016.

[18] I. Jacobson, *Object-oriented software engineering: A use case driven approach*. [New York] and Wokingham and Eng and Reading and Mass: ACM Press and Addison-Wesley Pub, 1992.

[19] A. Björkelund, B. Bohnet, L. Hafdell, and P. Nugues, “A High-performance Syntactic and Semantic Dependency Parser,” in *Proceedings of the 23rd International Conference on Computational Linguistics: Demonstrations*, 2010, pp. 33–36.