CREATE: A Co-Modeling Approach for Scenario-based Requirements and Component-based Architectures

Marcel Ibe, Martin Vogel, Björn Schindler and Andreas Rausch
Technische Universität Clausthal
Clausthal-Zellerfeld, Germany
{marcel.ibe, m.vogel, bjoern.schindler, andreas.rausch}@tu-clausthal.de

***Abstract*—Requirements engineering and architectural design are key activities for successful development of software-intensive systems. Both activities are strongly intertwined and interrelated. Particularly, in early development stages requirements and architecture decisions are frequently changing. Thus, advanced systematic approaches are needed, which could minimize the risks of wrong early requirements and architectural decisions. The fundamental problem addressed in this paper is the development of inconsistencies at the advanced approaches for co-evolution of requirements and architectures. Inconsistencies lead to an incorrect consideration of requirements by the system under development and consequently to unfulfilled requirements. In this paper, a domain specific model-based approach is presented, which supports the co-evolution of requirements and architectures. The approach provides simplified scenario-based models for the description of requirements, which are suitable for validation by stakeholders. Furthermore, the approach provides a component-based model for a precise and complete description of architectures. Adequate inter-relations between scenario-based and component-based models are defined, which support the consistence maintenance.**

***Keywords*-requirements; architecture; evolution; consistency.**

I. INTRODUCTION

Requirements Engineering (RE) and Architectural Design (AD) are essential for successfully developing high-quality software-intensive systems. RE and AD activities are intertwined and iteratively performed [2]. The architecture of a software system must satisfy its requirements, yet architectural constraints might prohibit certain requirements to be realized. This might imply a change to the initial requirements or the selection of a different appropriate architecture. Further, additional requirements might be discovered during the development process, leading to changes in the architecture. Design decisions that are made early in this iterative process are the most crucial ones, because they are very hard and costly to change later in the development process.

In classical development processes, artifacts like, for instance, the requirements specification or the architecture are developed sequentially. This is also the case at iterative process models like the spiral life cycle model of Böhm [1]. The iterative, concurrent evolution of requirements and architectures demands that the development of an architecture is based on incomplete requirements. Also, certain requirements can only be understood after modeling or even partially implementing the system architecture. Nuseibeh [2] describes an advanced approach, which adapts the spiral life cycle model and aims at overcoming the often artificial separation of requirements specification and design by intertwining these activities in an interactive evolutionary software development process. This approach is called the twin peaks model. To map requirements into architectures and maintaining the consistency and traceability between the two Grünbacher et al. [3] introduces an intermediate model called Component Bus System Property (CBSP) (see Fig. 1). This model maps requirements to architecture elements by the CBSP model, which allows a systematic way to reconcile requirements with stakeholders.

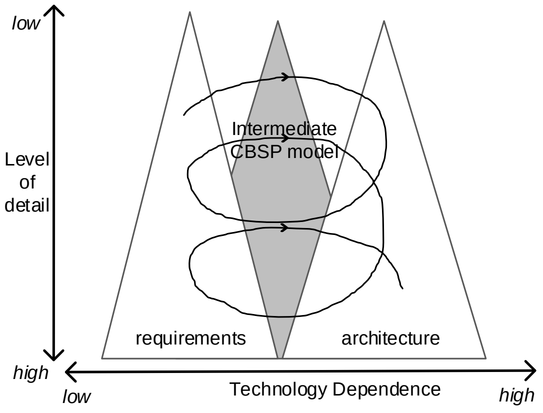

Figure 1. Intermediate model within the twin peaks [3]

Nevertheless, the advanced twin peaks model is kept very general. For instance, it does not specify the level of detail of requirements in relation to the architecture [4]. Due to the fact that there is no concrete advanced approach supporting the co-evolution of requirements and architecture we were commisioned by the German armed forces and the German government to undertake a research project. In order to be able to consider all required aspects, we made an expert survey. Therefore, we interviewed staff and leaders of three medium to big sized development projects with up to 30 project participants on customers and contractors side about their problems in the field of RE and AD.

A general mentioned problem was that the developed systems did not fulfill all requirements of the customers. The result of the survey was a list of the following reasons and derived guidelines:

- For the contractor the requirements were to informal, imprecise and incomplete. Requirements had to be repeatedly elicited and specified during architecture design. Hence, requirements on a software system should be complete and precise.
- At the elicitation process, the reconcilement of more precise and formal descriptions was to costly. The reason was the need of a detailed explanation by the contractor. For an improved reconcilement requirements descriptions should be precise as well as comprehensible. These guidelines are also mentioned by Nuseibeh [5]. Furthermore, the complexity of the models have to be manageable for validation by stakeholders.
- The most serious problem was caused by frequently changing requirements during architecture design. Changes frequently cause inconsistencies between requirements and architectures. Thus, requirements were frequently not fulfilled by the developed systems. Architectures have to describe how the system under development fulfills the given requirements by a precise definition of its structure and behavior. Consistency constraints between requirements and architectures should be defined, which enable an automatic support of the consistence maintenance.

Starting from this initial situation the target of the project was the development of a domain specific model-based approach, which fulfilles the mentioned guidelines. The subject of this paper is the presentation of the developed domain specific co-modeling approach CREATE for the description of requirements and architectures. Furthermore, the experiences at the application in practice are described. The presented approach is domain specific for interactive information systems like web-based systems and modern communication systems. In Section II, existing model-based approaches for the co-evolution of requirements and architectures are considered. In Section IV, the overall approach is introduced and in Section V the approach is shown at an example. Section VI contains a description of our experiences at the development and application of the approach in practice. Section VII includes a discussion of the results and pending points for future work.

II. Related Work

Existing model-based development approaches for requirements and architectures can be categorized into model-based approaches for requirements engineering, model-based approaches for architecture design and combined approaches.

Representative model-based approaches for requirements engineering are described in [6]–[8]. In [6], requirements are described by Unified Modeling Language (UML) [18] activity diagrams. A formal operational semantics enables execution of activity diagram specifications. The executed activity diagram specification serves as prototype for visualization of requirements. In the approach illustrated in [7], UML collaboration diagrams are enriched by user interface information in order to specify elicited requirements. These diagrams are transformed into complete dynamic specifications of user interface objects represented by state diagrams. These state diagrams are used for generation of prototypes. In [8], use case and user interface information are recorded at stakeholder interviews. Therefore, use case steps are enriched by scribbled dialog mockups. Prototypes are created, which visualize dialog mockups of use case steps in sequence for fast feedback of stakeholders. In general, these approaches have a well elaborated model structure for requirements engineering and improve the validation of requirements by stakeholders. On the other side, the mapping to the architecture is not precisely enough defined at these approaches to support a co-evolution of requirements and architectures.

Representative model-based approaches for architecture design are described in [9,10]. In Model-Driven Architecture (MDA) [9], the Computation Independent Model (CIM) can be used to describe business processes. The Platform Independent Model (PIM) may describe the structure and behavior of the software system. Component models like KobrA [10] are concrete approaches supporting MDA. In general, these approaches have a well elaborated model structure for architecture design and enable a detailed description of the structure and behavior of the software system. On the other side, these approaches do not support a co-evolution of requirements and architectures. The mapping between requirements and architectures is not precisely enough defined for this field of application.

Representative combined modeling approaches for requirements and architectures are described in [3,11,12]. In [11], a Requirements Definition Language (RDL) is used, which allows a structured definition of requirements. Meta model elements of the RDL are mapped to corresponding meta model elements of the Architecture Description Language (ADL). The approach described in [3] uses the intermediate model CBSP to map requirements to architecture elements. Different subtypes of CBSP elements allow classification of requirements. Requirements exhibit overlapping CBSP properties can be split and refined until no stakeholder conflicts exist. The Software Architecture Analysis Method (SAAM) [12] describes a method for a scenario-based analysis of software architectures. In this method, scenarios and architecture descriptions are developed iteratively. For each scenario it is determined whether a change of the architecture is required for execution. Based on the importance and conflicts of required changes an overall ranking of the developed scenarios is determined.

An advantage of these approaches is the combination of models for the description of requirements and architectures. On the other side, these approaches are very abstract and do not specify concrete models and mappings, which fulfill the conditions defined in the introduction for an adequate description of requirements and architectures.

Besides the stated existing approaches further approaches are conceivable, which are based on synthesis approaches [13] of complete state-based models from scenario-based models. Scenario-based and state-based models can potentially be used for the description of requirements and architectures. Consistency is, for instance, a subject of the approaches described in [14,15]. Unfortunately, these approaches are generally maintaining a complete consistency by means of a bijection. Architectures need to describe more details about the software system. These details have to be well separated from the requirements. Hence, an alternating correction of inconsistencies and not a bijection is required for the support of a co-evolution of requirements and architectures.

III. Contribution

The main contributions of this paper can be summarized as follows:

- Definition of a domain specific model-based approach for requirements engineering and architecture design in the sense of twin peaks. Requirements descriptions have to be precise and comprehensible. This necessitates a well-balanced trade-off between expressiveness and manageability of models for the description of requirements. Furthermore, the architecture has to provide a detailed description of the behavior and the resulting structure of the software system. In our domain specific approach, simplified scenario-based requirements models are defined for the description of requirements and component-based models for the description of architectures.
- Definition of consistency constraints which support a co-evolution of requirements and architectures. Complex dependencies between requirements and architectures cause a high complexity for consistence maintenance. Thus, not only for the requirements model, but also for the inter-relations between requirements and architecture models a well-balanced trade-off between expressiveness and manageability is necessary. In our approach, inter-relations between scenario-based requirements and component-based architecture models are provided, which enable an automatic consistence maintenance.
- This paper presents the results of the evaluation of the approach at real system development projects. Several iterations of phases for model definition and practice tests were required to find the presented solution. The approach is presented by a case study.

IV. Overall Approach

Our domain specific model-based approach supports concurrent development of requirements and architectures. An appropriate process for concurrent development is described by the twin peaks model [2]. In this model, requirements and architectures have an equal status and are evolved iteratively. This is illustrated by twin peaks (see Fig. 2).

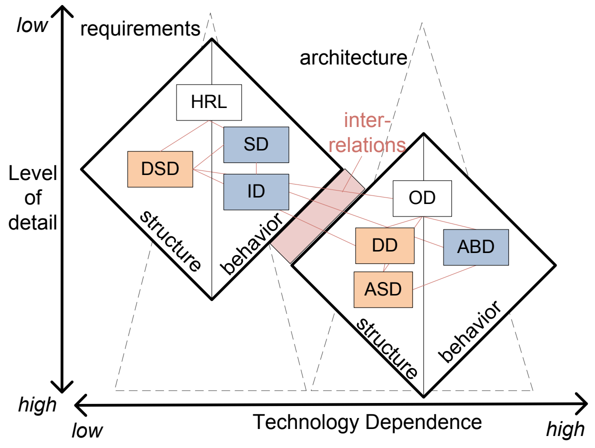

Figure 2. Co-modeling approach within twin peaks

Our domain specific model-based approach concretizes twin peaks by defining a concrete description technique. Diagrams are used for a precise description of requirements and architectures. These diagrams are illustrated within diamonds in the twin peaks model (see Fig. 2). The process flow of our approach begins with a formal description of inital requirements. Afterwards, the architecture is developed and consistence to the requirements is maintained continuously. Inconsistencies are resolved by changing requirements or the architecture.

The main contribution of our domain specific approach is the concrete description technique with well-defined inter-relations between requirements and architecture descriptions. It is well known that scenarios help to elicit and validate requirements [13]. A precise description of elicited requirements can be achieved by scenario-based models [13]. The co-modeling approach provides simplified scenario-based models for the description of requirements. Furthermore, the description is reduced to representative and concrete scenarios. Hence, the complexity of these models is manageable for the validation by stakeholders. The validation is improved by combining these models with models enabling visualization of requirements by user interface mockups [8]. During AD, the architecture of the co-modeling approach is developed. An architecture describes the behavior and the resulting structure of the software system precisely. This description is enabled by a component-model. Component-Based Software Engineering (CBSE)

[16] has been continuously improved and successfully applied over the past years. Systems are composed by existing software 'parts' called software components. Component models enable a precise description of component-based architectures [17].

In our domain specific model-based approach, diagrams are used to model structural or behavioral aspects of requirements or architectures. For instance, elicitation and specification of processes at the domain (e.g., business processes) is an important aspect at requirements engineering. In our approach, these processes can be described by a Scenario Diagram (SD). Thus, the SD is assigned to the behavior part of the requirements diamond (see Fig. 2). In Section V, the provided diagrams and their inter-relations are described in detail. Some models, for instance, the Hierarchical Requirements List (HRL) can be used to describe structural as well as behavioral aspects. Existing languages, such as UML [18], include among others structural and behavioral diagrams for the modeling of systems. Our domain specific approach uses exemplarily a subset of UML diagrams and their available model elements to formally describe requirements and architectures. Additional models are used to enable a visualization of requirements by user interface mockups.

Consistence maintenance during the development of requirements and architectures is supported by well-defined inter-relations between scenario-based requirements models and component-based architecture models (see Fig. 2). Inter-relations are also defined within these models. They are defined by associations between model elements and additional consistency constraints. The defined inter-relations enable an automatic consistence maintenance by, for instance, checking the consistency constraints and permitting changes not until detected inconsistencies are solved.

V. Modeling Example

Details of the description technique and the consistence constraints of the domain specific model-based approach are shown at a case study. The subject is the development of a library system.

A. Scenario Description

1) HRL: The HRL enables a text-based description of structural and behavioral requirements. They can be arranged hierarchically. In this way, it is possible to refine one requirement by several other requirements. In our example, the HRL contains some structural information about the system environment. The requirements list describes the system under development, the user of the system and an entity that should be managed by the system (red marked in Fig. 3 upper left). The requirement *show statistics* describes a desired behavior of the system. *Manage books* is a very general requirement and is refined by the requirement *show statistics*, which is more precise.

2) Domain Structure Diagram (DSD): The domain structure, e.g., the business structure, can be described by the DSD. It is based on UML Composition Structure Diagrams [18]. First, the domain structure consists of systems and persons as well as their ability to communicate to each other described by parts and connections of the DSD. The DSD *Library* describes the system to develop, the library system and a person, the employee (see Fig. 3 requirements left). The connection between the employee and the library system assumes that they can interact with each other. Furthermore, the DSD describes the relevant entities by parts. Currently, there is only one of the type *Book*. The parts of the domain structure (e.g., persons) have to be initially mentioned in the HRL (see gray line in Fig. 3).

3) SD: The description of processes at the domain (e.g., business processes) is an important task at requirements engineering. Processes can be described precisely by the SD, which is based on scenario-based UML Communication Diagrams [18]. SD describes representative scenarios at the domain, which have to be supported by the system under development. The scenarios are described as a sequence of messages between instances of the parts introduced in the DSD. Messages between two instances can only be sent if a connection exists between the corresponding parts of the DSD. In our example, the scenario *ShowBookStatistic* describes an interaction between an employee *m* and the library system with two messages (see gray line in Fig. 3 requirements upper right).

4) Interaction Mockup Diagram (ID): The ID can be used to visualize requirements to stakeholders. For this, it describes the messages of the SD by interaction mockups. Interaction mockups are user interface mockups, which visualize interactions of the system in general. In the SD, it is possible to describe the source and target of a message. In the ID, every message is detailed by exactly one interaction mockup for visualization. The scenario can be visualized to stakeholders by showing the interaction mockups step by step following the sequence of the messages of the SD. In our example, one interaction mockup shows the summary of all books of the message *1* in SD *ShowBookStatistic* (see gray line in Fig. 3 requirements). Another shows the detailed view of one book with its statistic.

B. Architecture Design

An architecture is a plan, which describes how a software system fulfills given requirements. The following architecture models allow a complete description of the behavior and the resulting structure of the system under development.

1) System Overview Diagram (OD): The OD of the architecture is based on the UML Use Case Diagram [18]. It is used to describe the most abstract structure and behavior of the system and its context by the system boundary and the associated use cases, which are called functions. The actors and the system context are derived from the DSD,

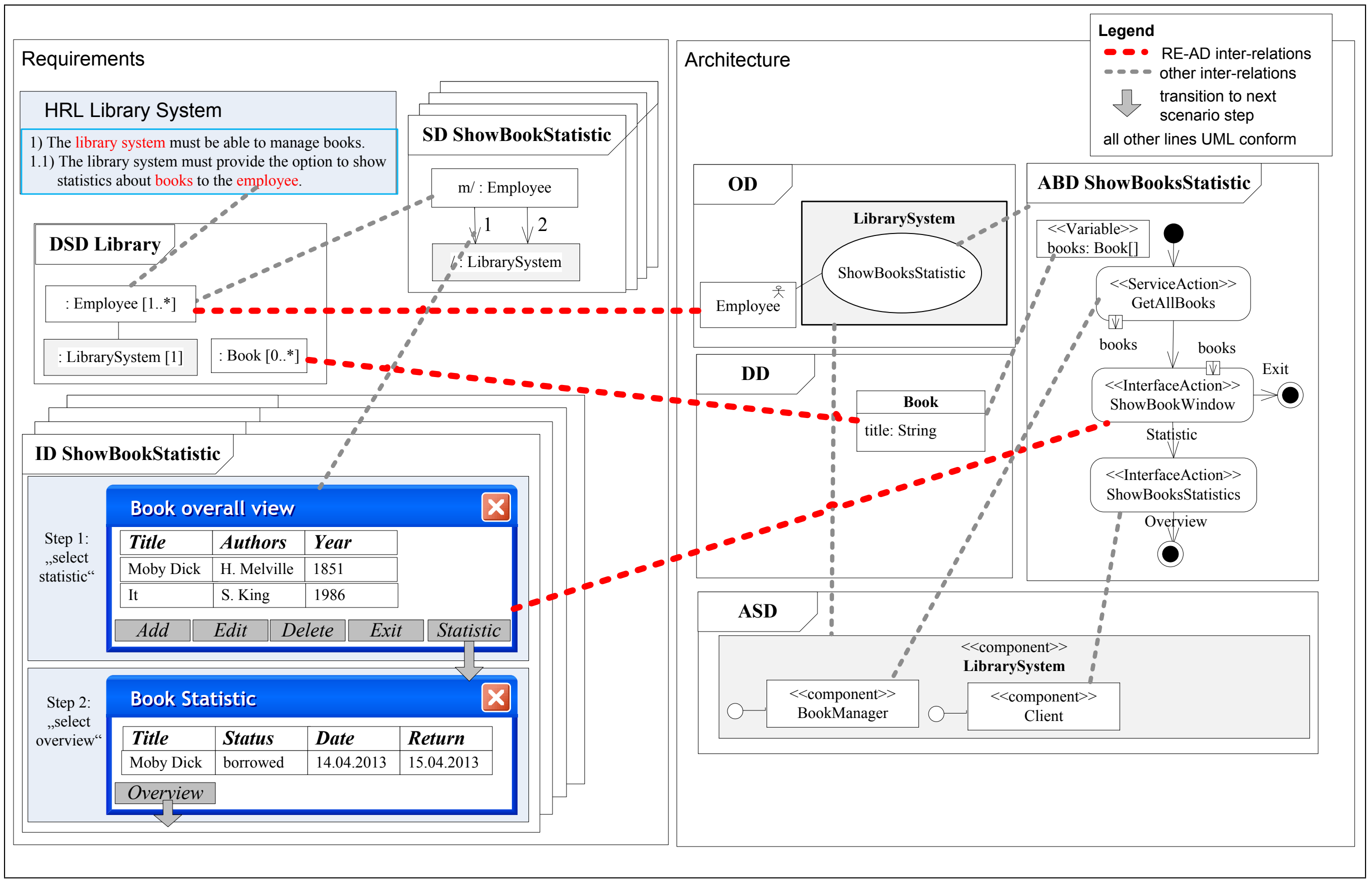

Figure 3. Requirements models, architecture models and inter-relations

the functions from SD. The OD of the example describes a system with the function *ShowBooksStatistics* and the actor employee (see Fig. 3 architecture upper right). Employee is connected to the function. This connection also exists between DSD (Employee) and SD (ShowBookStatistic). The process of a function is described by an architectural behavior diagram.

2) Architectural Behavior Diagram (ABD): The ABD describes the behavior of the software system and is based on the UML Activity Diagram including data flow. The ABD describes the process of the functions defined in OD completely. The function *ShowBookStatistic* is, for instance, described by the activity *ShowBookStatistic* (see gray line in Fig. 3 right). Within the ABD different action types like *InterfaceAction* and *ServiceAction* are used. A *ServiceAction* is performed by the system (e.g., a database call). An *InterfaceAction* describes an interaction of the system with its environment and is, therefore, associated with an interaction mockup of the ID. The action *ShowBooksStatistics* is, for instance, associated with an interaction mockup (see Fig. 3 right).

3) Data Diagram (DD): At a function described by ABD data objects can be used by the system. The DD is based on UML Class Diagrams [18] and describes the data types of the data objects. For example, the DD describes a type *book*, which is the type of the variable *books* of the ABD *ShowBooksStatistics* (see gray line in Fig. 3 architecture left). The data objects to be processed by the system are derived from the entities of DSD. The DD describes these entities in more detail. If there is a connection between two parts within the DSD, a relation must exist between the types of the parts and the corresponding data types in DD.

4) Architectural Structure Diagram (ASD): The ASD is based on UML Component Diagrams [18] and describes the internal components of the system under development and their offered interface as a black-box view. Subsequently, the components are further decomposed to refine their internal structure. The ASD *LibrarySystem* describes, for instance, the internal structure of *LibrarySystem* of the OD (see gray line in Fig. 3 architecture bottom). The internal structure is derived from the actions of the ABD. Hence, each component must be associated with an action of an ABD. The component *LibrarySystem* is refined by a component *BookManager*, which is associated with the action *GetAllBooks* of the ABD as well as the component *Client*.

C. Consistence constraints

The consistence maintenance is supported by the definition of inter-relations between requirements and architecture models. Inter-relations between models of the scenario-based requirements and the component-based architecture (e.g., between HRL and DSD) are described in Section V-A and V-B (see gray lines in Fig 3). Essential for the concurrent development are the inter-relations between requirements and architectures (see red lines in Fig 3). Inter-relations are defined in the term of associations and additional consistency conditions. A violation of a consistency condition means an inconsistency. We defined all necessary inter-relations. In the following, a subset of these inter-relations are exemplarily introduced, which is most suitable to explain the dependencies between our requirements and architecture descriptions:

- (1) The existence of an entity in the DD implies the existence of a corresponding type in the DSD.
- (2) The existence of a type in DSD whose part is directly connected with the part of the system to build implies the existence of a corresponding actor in the OD.
- (3) Every interaction mockup in the ID must be mapped on exactly one *InterfaceAction* in an ABD.

Probable changes during the development of concurrently evolved requirements and architectures are illustrated in Fig 4. While modeling the architecture it was noticed, that the system has not only to handle books. Also magazines should be managed. Hence, a new entity *Magazine* was added to the DD. As a consequence, the entity Media, as a generic term was introduced. Respectively two new inheritance relations were added. After this, the consistency condition (1) is violated. The DSD doesn't contain any corresponding element to these two new entities from the DD (arrows (1) in Fig. 4). A change in the requirements model was necessary, when it became clear that the manager needs other statistics about a book, then an employee. Hence, the manager as a new part of the system environment is added. In consequence, the condition (2) is violated. The manager is directly connected to the system but there is no corresponding actor in the OD (arrow (2) in Fig. 4). Because of the new needs of the manager, the scenario also has to be adapted. Depending on who uses the system, the shown information about a book varies. Thus, a new interaction mockup for the manager has to be added. This violates again the condition (3). The new interaction mockup is not mapped on an InterfaceAction from the ABD (arrow (3) in Fig. 4).

To correct these inconsistencies, a few further changes have to be made. It is necessary to add the entities *Media* and *Magazine* to the DSD. After this consistency condition (1) holds again (see arrows (1) in Fig. 5). To comply with the second condition, a new actor for the manager has to be introduced into the OD (arrow (2) in Fig. 5). Finally, a mapping from the added interaction mockup to an InterfaceAction is missing. One could map the new interaction mockup to an existing InterfaceAction or extend the ABD by a new InterfaceAction. By extending the ABD by the InterfaceAction *ManagerStats* the interaction mockup can be mapped on it (arrow (3) in Fig. 5). The new action may be processed by a new component *ManagerClient* at the ASD. By making these changes all consistency conditions were restored. As shown above, checking the consistency conditions helps to detect inconsistencies. An automatic support of the consistence maintenance can, for instance, be realized by permitting changes not until all inconsistencies are solved.

VI. Evaluation

The development of the co-modeling approach took place at research projects in cooperation with a public institution over a period of four years. At these research projects, we gave advice and supported to system development projects in order to test our results in practice. The goal of the overall approach is to support consistence maintenance of requirements and architectures in early development phases. The goal of the evaluation was to test the usability and the inconsistence prevention of our approach. At a first step, we developed the component-based architecture model for a precise description of the architecture. For reconcilement with stakeholders we developed a prototype generator, which is able to interpret the developed models. The stakeholders should validate the architecture and the consistence to their requirements with the aid of the prototypes. This approach was tested at a system development project over a period of one year. The subject of this project was a communication system. At this project, a model was developed comprising 20 system functions, 253 activity nodes and 35 data types. Conclusive it revealed that the usability of the approach has to be improved. The number of possible states described by the component-based architecture leads to less comprehensibility to stakeholders. They were not able to agree to the developed specifications. Consequently, the consistence maintenance of requirements and architectures could not be supported by this approach. Based on the results of this practice test we extended the approach by scenario-based models. This extended co-modeling approach, which is introduced in this paper, was tested in practice at a further system development project with a similar subject over a period of one year. In this period, the usability was significantly better. Stakeholders were able to agree to the visualized and scenario-based requirements. Furthermore, they were able to give helpful feedback, which leads to a big number of changes. We measured at three milestones the number of changes, the detected errors and especially remaining inconsistencies. Between these milestones we documented 500 changes and 67 errors. 8 of these errors were inconsistencies. The rate of inconsistencies to changes is low. For an indication, at a study described in [19], change

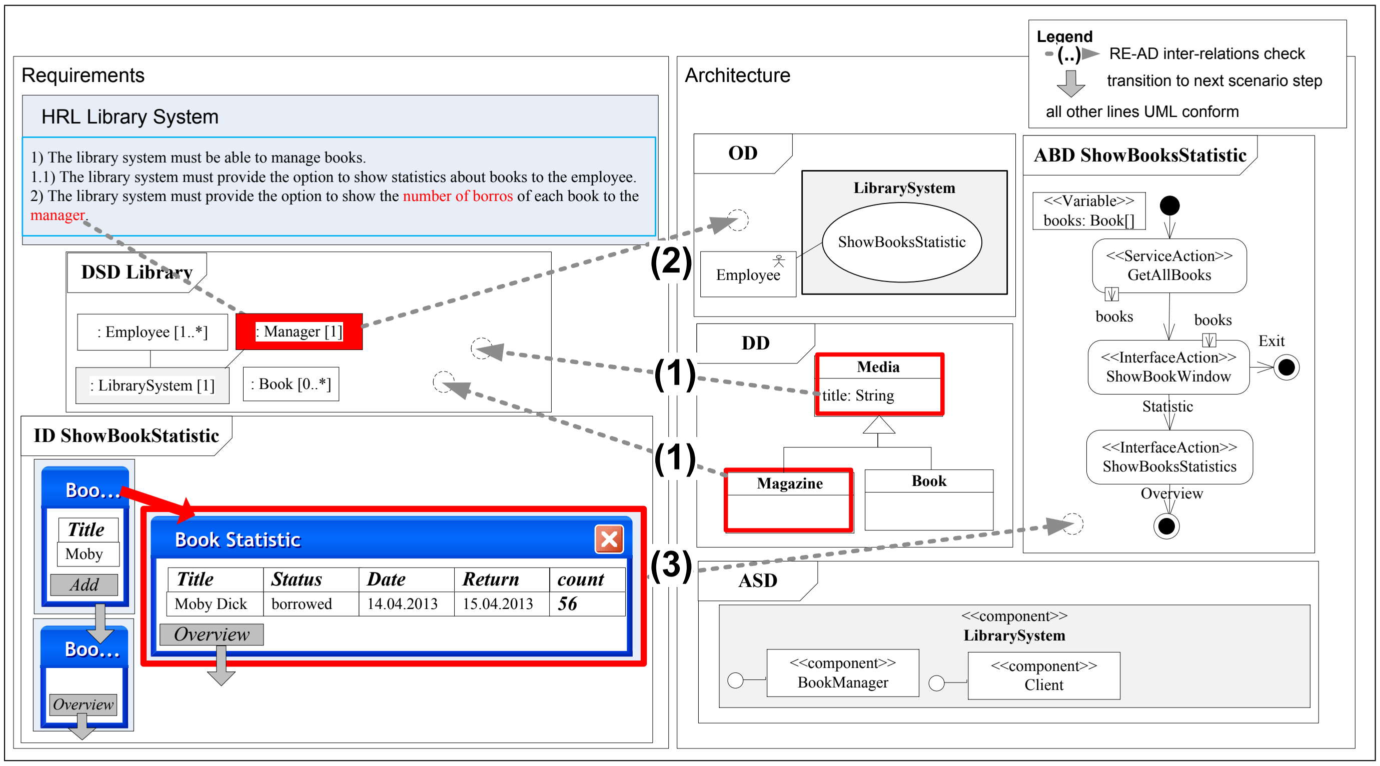
Legend
(..) RE-AD inter-relations check
transition to next scenario step
all other lines UML conform
Requirements
HRL Library System
1) The library system must be able to manage books.
1.1) The library system must provide the option to show statistics about books to the employee.
2) The library system must provide the option to show the number of borros of each book to the manager.
DSD Library
: Employee [1..*]
: Manager [1]
: LibrarySystem [1]
: Book [0..*]
ID ShowBookStatistic
Boo...
Title
Moby
Add
Overview
Book Statistic
Title
Status
Date
Return
count
Moby Dick
borrowed
14.04.2013
15.04.2013
56
Architecture
OD
LibrarySystem
ShowBooksStatistic
Employee
DD
Media
title: String
Magazine
Book
ASD
<<component>> LibrarySystem
<<component>> BookManager
<<component>> Client
ABD ShowBooksStatistic
<<Variable>> books: Book[]
<<ServiceAction>> GetAllBooks
books
books
Exit
<<InterfaceAction>> ShowBookWindow
Statistic
<<InterfaceAction>> ShowBooksStatistics
Overview
(1)
(2)
(3)

Figure 4. Changes at the requirements and architecture model

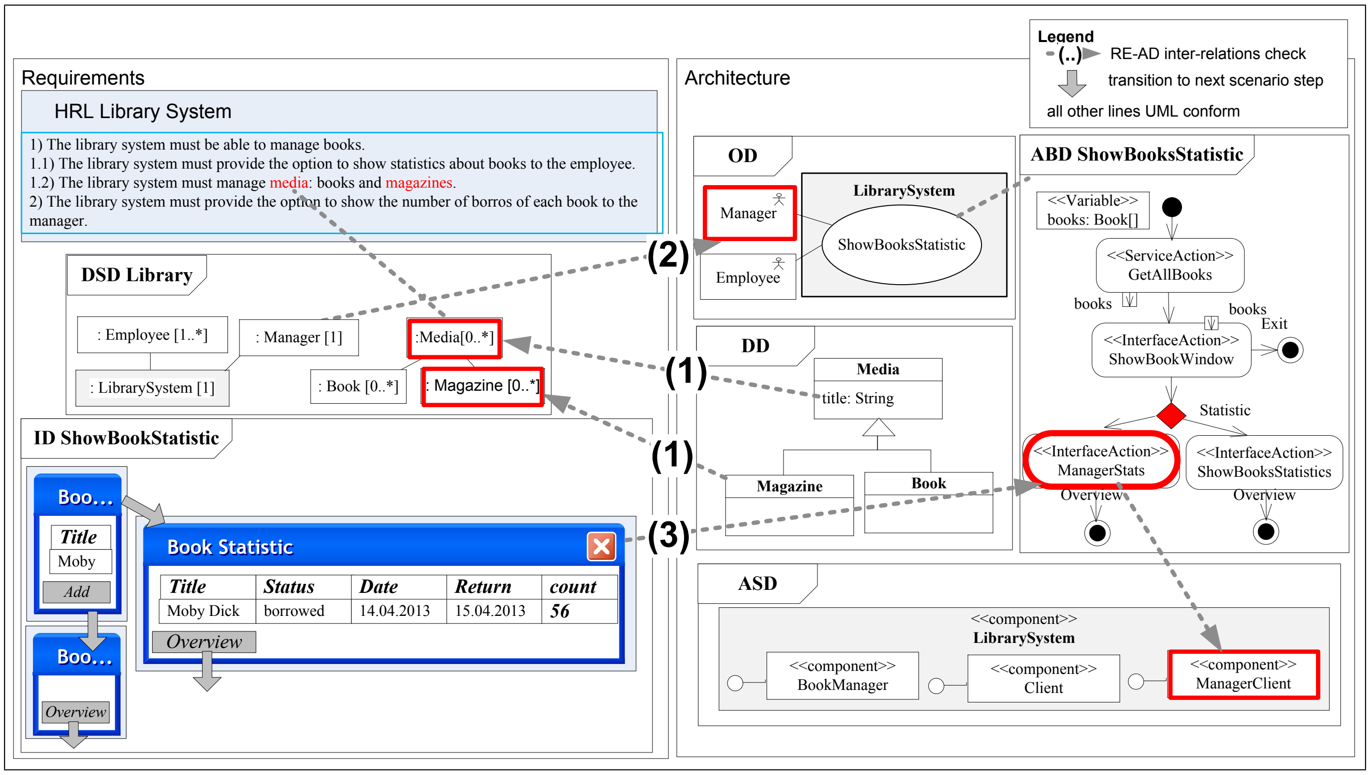
Legend
(..) RE-AD inter-relations check
transition to next scenario step
all other lines UML conform
Requirements
HRL Library System
1) The library system must be able to manage books.
1.1) The library system must provide the option to show statistics about books to the employee.
1.2) The library system must manage media: books and magazines.
2) The library system must provide the option to show the number of borros of each book to the manager.
DSD Library
: Employee [1..*]
: Manager [1]
:Media[0..*]
: LibrarySystem [1]
: Book [0..*]
: Magazine [0..*]
ID ShowBookStatistic
Boo...
Title
Moby
Add
Overview
Book Statistic
Title
Status
Date
Return
count
Moby Dick
borrowed
14.04.2013
15.04.2013
56
Architecture
OD
Manager
LibrarySystem
ShowBooksStatistic
Employee
DD
Media
title: String
Magazine
Book
ASD
<<component>> LibrarySystem
<<component>> BookManager
<<component>> Client
<<component>> ManagerClient
ABD ShowBooksStatistic
<<Variable>> books: Book[]
<<ServiceAction>> GetAllBooks
books
books
Exit
<<InterfaceAction>> ShowBookWindow
Statistic
<<InterfaceAction>> ManagerStats
<<InterfaceAction>> ShowBooksStatistics
Overview
Overview
(1)
(2)
(3)

Figure 5. Changes to solve the inconsistencies

data of requirements documents are analyzed. In this study, 88 changes, 79 errors, and 16 inconsistencies were detected.

VII. Conclusion and Future Work

The fundamental problem addressed in this paper was the development of inconsistencies at the advanced approaches for co-evolution of requirements and architectures. In this paper, a domain specific model-based approach was introduced, which supports a co-evolution of requirements and architectures. The approach uses a scenario-based model for a precise description of requirements and a component-based model for the description of architectures. Well-defined inter-relations enable an automatic consistence maintenance.

A frequently stated argument is the entailment of high costs for the development of precise requirements and architecture models at a software project. This can be countered by the fact that an incorrect consideration of requirements not uncommonly leads to complete project failures. Thus, maintaining the consistency at the co-evolution of requirements and architectures is important. Supporting this task by models enabling an automatic consistence maintenance reduces the risk of a project failure and costs for consistence maintenance. Furthermore, the developed models can be reused for automatic generation of code, test cases and documents like, for instance, requirements specifications. Nevertheless, the usage of formal models at a development project should, among others, be made conditional on the size of the project. At the beginning of a development project, the advantages and disadvantages of using formal models have to be weighed.

As future work, a further evaluation is planned to compare the effectivity of the co-modeling approach to other model-based approaches for requirements and architectures. Furthermore, it is planned to develop a tool for automatic consistence maintenance.

References

[1] B.W. Böhm, ”A spiral model of software development and enhancement”, IEEE Computer Society Press, vol. 21, May 1988, pp. 61–72.

[2] B. Nuseibeh, ”Weaving Together Requirements and Architectures”, IEEE Computer Society Press, vol. 34, March 2001, pp. 115–117.

[3] P. Grünbacher, A. Egyed, E. Egyed, and N. Medvidovic, ”Reconciling Software Requirements And Architectures With Intermediate Models” in Software and Systems Modeling. Springer, 2003, pp. 202–211.

[4] R. Ferrari and N. H. Madhavji, ”The Impact of Requirements Knowledge and Experience on Software Architecting: An Empirical Study” in Working IEEE/IFIP Conference on Software Architecture (WICSA), 2007, pp. 44–54.

[5] B. Nuseibeh and S. Easterbrook, ”Requirements Engineering: a roadmap” in Proceedings of the Conference on The Future of Software Engineering (ICSE), ACM Press, 2000, pp. 35–46.

[6] C. Knieke and U. Goltz, ”An executable semantics for UML 2 activity diagrams” in Proceedings of the International Workshop on Formalization of Modeling Languages (FML), ACM Press, 2010, pp. 3:1–3:5.

[7] M. Elkoutbi, ”Automated Prototyping of User Interfaces based on UML Scenarios” in Journal of Automated Software Engineering, vol. 13, Kluwer Academic Publishers, 2006, pp. 5–40.

[8] K. Schneider, ”Generating fast feedback in requirements elicitation” in Proceedings of the 13th international working conference on Requirements engineering: foundation for software quality (REFSQ), Springer-Verlag, 2007, pp. 160–174.

[9] A. G. Kleppe, J. Warmer, and W. Bast, ”MDA Explained: The Model Driven Architecture: Practice and Promise”, Addison-Wesley Longman Publishing Co. Inc., 2007

[10] C. Atkinson, J. Bayer, and D. Muthig, ”Component-Based Product Line Development: The KobrA Approach” in Software Product Line Conference, Denver, Kluwer Academic Publishers, 2000, pp. 289-309.

[11] R. Chitchyan, M. Pinto, A. Rashid, and L. Fuentes, ”COMPASS: Composition-Centric Mapping of Aspectual Requirements to Architecture” in Transactions on AspectOriented Software Development, 2007, pp. 3–53.

[12] R. Kazman, G. Abowd, L. Bass, and P. Clements, ”Scenario-Based Analysis of Software Architecture” in IEEE Softw., vol. 13, IEEE Computer Society Press, Nov. 1996, pp. 47–55.

[13] H. Liang, J. Dingel, and Z. Diskin, ”A comparative survey of scenario-based to state-based model synthesis approaches” in Proceedings of the 2006 international workshop on Scenarios and state machines: models, algorithms, and tools (SCESM), ACM Press, 2006, pp. 5–12.

[14] Y. Bontemps, P. Schobbens, and C. Löding, ”Synthesis of Open Reactive Systems from Scenario-Based Specifications” in Proceedings of Application of Concurrency to System Design, 2003, pp. 41–50.

[15] V. Garousi, L. Briand, C. Lionel, and Y. Labiche, ”Control Flow Analysis of UML 2.0 Sequence Diagrams” in Model Driven Architecture Foundations and Applications, 2005, pp. 160–174.

[16] C. Szyperski, ”Component Software: Beyond Object-Oriented Programming”, Addison-Wesley Longman Publishing Co. Inc., 2002.

[17] A. Rausch, R. Reussner, R. Mirandola, and F. Plasil, ”The Common Component Modeling Example: Comparing Software Component Models”, ser. Springer Lecture Notes in Computer Science, vol. 5153, 2008.

[18] OMG, ”UML, Version 2.2. OMG Specification Superstructure and Infrastructure”, 2009.

[19] V. R. Basili and D. M. Weiss, ”Evaluation of a software requirements document by analysis of change data” in Proceedings of the 5th international conference on software engineering, IEEE Press, 1981, pp. 314–323.