Methodology for Requirements Analysis and Design in Development of Service-Oriented Information Systems

Lina Ceponiene, Lina Nemuraite, Jonas Ceponis

Kaunas University of Technology, Studentu 50-315a, LT 51368 Kaunas, Lithuania,
lina.ceponiene@ktu.lt, lina.nemuraite@ktu.lt, jonas.ceponis@ktu.lt

Abstract

Service orientation and Model Driven Development are the pillars of modern Information Systems. Service-oriented information systems are composed of loosely coupled and interoperable services driven by requirements of users. New modelling facilities are proposed to enable effective service development including more rigorous requirements definition, reconciliation and transformations of requirements to software architectures. These facilities are focused on modelling interactions and state transitions of services represented by interfaces and entities of problem domain. Metamodels for requirements and design model specification, algorithms for requirement reconciliation and transformation to design were created using UML and OCL, and implemented in UML CASE tools.

Keywords: service-oriented, information system, requirements, MDD, MDA, UML, OCL.

1. Introduction

During development of modern information systems the extensive complex of technologies is required, which would be difficult to tackle without the automation tools. Requirements for software applications are constantly changing and thus the development of software product continues to the end of its lifecycle. Furthermore, the considerable part of the system under development encompasses schemas, specifications, and program code, which can be automatically generated. Service-oriented development would be ineffective without Model Driven automation.

Consequently, the Model Driven Development (MDD) methods gain in popularity. These methods enable system design at more abstract – model – level and program code generation from models, thus transforming application development to model

development. The Object Management Group (OMG) organization manages Model Driven Architecture (MDA) [Frankel (2003)], which is one of MDD methodologies. Presently MDA concentrates on the last stages of development process and automates transformations from Platform Independent Model (PIM) to Platform Specific Models (PSM), and to program code. Initial development stages which encompass business modelling, requirements analysis and specification, and domain modelling, are not analyzed in detail.

Generation of meaningful program code of high quality is possible only if there is a comprehensive and consistent information system model which describes its structure and behaviour. This model should be defined in initial development stages, independently not only from implementation platform (as PIM) but also from logical application architecture. Thus such model would be adjustable not only to various implementation platforms but also to various architectural decisions. In this work, such model was named Design Independent Model (DIM) [Ceponiene et al. (2003)], [Ceponiene et al. (2005a)], [Ceponiene et al. (2005b)], [Ceponiene et al. (2005c)]. It is a detail requirements model comprehensively describing structure and behaviour of information system under development. DIM differs from design model in the fact that there are no design decisions in DIM – it does not include any control classes or components. Proposed DIM consists of entities of problem domain and interfaces to behaviour defining the sets of conceptual operations, constituting the basis for service-oriented models.

The rest of the paper is organized as follows. In section 2 the related work is analysed. Section 3 presents the overview of the proposed method. The first stage of the method – initial requirements definition – is discussed in section 4. In section 5, the structure of Design Independent Model is defined. Section 6 discusses the problem of reconciliation of requirements. In Section 7 service design pattern and transformation from requirements to design is described. Section 8 discusses the implementation of the proposed method and finally, section 9 draws some conclusions.

2. Related Work

Recently, OMG has issued the RFP for UML Profile and Metamodel for Services [OMG (2006)], where requirements are stated to “complement existing UML metamodel by defining an extension to UML to ensure complete and consistent service specifications and implementations“. Our work addresses a part of OMG requirements by concentrating on service-oriented modelling in the early stages and linking it with Model Driven Development (Fig.1): modelling and reconciliation of requirements and transition from requirements to design. In MDD methods neither specification of requirements model, nor another important step – transformation from requirements to design – is clearly defined.

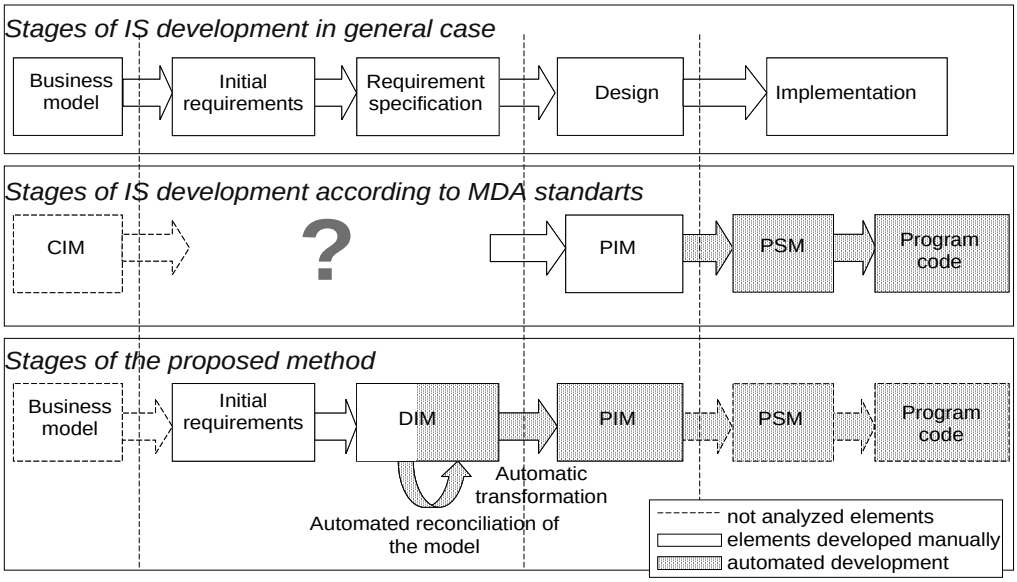

Figure 1. *Generic and MDA Stages of IS Development in Comparison with Our Work*

Existing methods of information system development were analyzed for finding solutions to the problems encompassing specification of requirements model, its reconciliation and transformation to design model. Ideas of comprehensive requirements modelling were based on [Astesiano et al. (2002)]. But rules for transition from requirements to design models are not identified in this method. Rational Unified Process [Jacobson et al. (1999)] is a very wide development process, which accentuates iterative development and separation between requirements and design models. But due to its volume, RUP can be used only after the process of adaptation, which is similar to creating your own method, based on RUP. In opposite to RUP, eXtreme Programming [Beck (1999)] accentuates fast development and using as small amount of models as possible. ICONIX [Rosenberg et al. (2001)] is somewhere between RUP and XP, it uses a relatively small amount of models and offers a fast transition from requirements to code. But reconciliation of the system model is not employed in ICONIX and the stages of requirements specification and design are not strictly separated. Such separation is clearly defined in Catalysis [D'Souza et al. (1999)] development process which also focuses on component based development. MERODE [Dedene et al. (1994)], an event driven development method, separates requirements and design models, and defines formal rules for transition from one model to another, but it does not approach users and interactions. All of the explored methods do not automate initial development stages analyzed in our work.

MDA uses the Unified Modelling Language (UML) [OMG (2003a)]. UML is widely accepted standard used for IS development, but there is a problem in reconciliation of

UML models. A UML model of the system consists of several models – diagrams of different kinds, expressing properties of different aspects of a system. The meaning of each diagram kind can be given in isolation, but nevertheless each diagram describes the view of the overall system. Many diagrams and their elements are overlapping and represent the same things from different viewpoints. Existing universal CASE tools do not have any means for definition of comprehensive requirements specification, for its reconciliation and transformation into design models. There still exist a lot of possibilities for improvement of IS development processes in CASE tools and increase of the degree of automation of these processes.

In presented method semantics of information system is defined according to the semantics of Labelled Transition Systems [Reggio et al. (2001)]. In UML, such information system is presented as hierarchical state machine composed of interacting state machines; state transitions here are triggered by external events and execution of these transitions is constrained by rules dependent on possible states of entities. The consistency of system model described in UML and OCL [OMG (2003b)] may be ensured by integrating interactions and state transitions, and also by interconnecting states of behavioural objects (services) and state describing objects (entities). This part of the method is partially associated with the algorithms for generating state machines from sequence diagrams [Makinen et al. (2000)], [Whittle et al. (2000)], but these algorithms ensure only unidirectional transformation, whereas in our work the algorithms for bidirectional transformations between sequence and state diagrams are defined.

Design model is obtained from requirements model DIM by applying the architectural pattern created on the base of [Gamma et al. (1995)]. During transformation from requirements to design, DIM specification is allocated to elements of the pattern.

In service-oriented information systems two main concepts must be modelled: entities and services. A possible way for implementing such system is the Web Services Architecture. Due to the limits of size of this paper, the relation of our method with abundant standards for services and MDA is not presented. In service-oriented systems automation gives an obvious effect: the same conceptual model may be used for generating program code, database and XML schemas, services specifications in WSDL, etc. In the proposed method this list is expanded: it is also possible to generate design model and ensure the consistency of this model.

3. The Method Overview

Process of development of information system, using DIM, may be defined as a sequence of transformations between models: $DIM \xrightarrow{T1} PIM \xrightarrow{T2} PSM \xrightarrow{T3} I$, here I indicates implementation. Process investigated in this work is development of DIM and going from DIM to PIM: $DIM \xrightarrow{T1} PIM$, where integrity must be ensured for DIM, PIM and

between DIM and PIM. The integrity means consolidation of overall structure and behaviour of the system, represented in different views, expressed in UML use case, class, interaction diagrams and state machines. The proposed method for Service-oriented Information System Requirements Analysis and Transformation to Design consists of three main stages: initial requirements specification, DIM development, and generation of PIM (Fig.2).

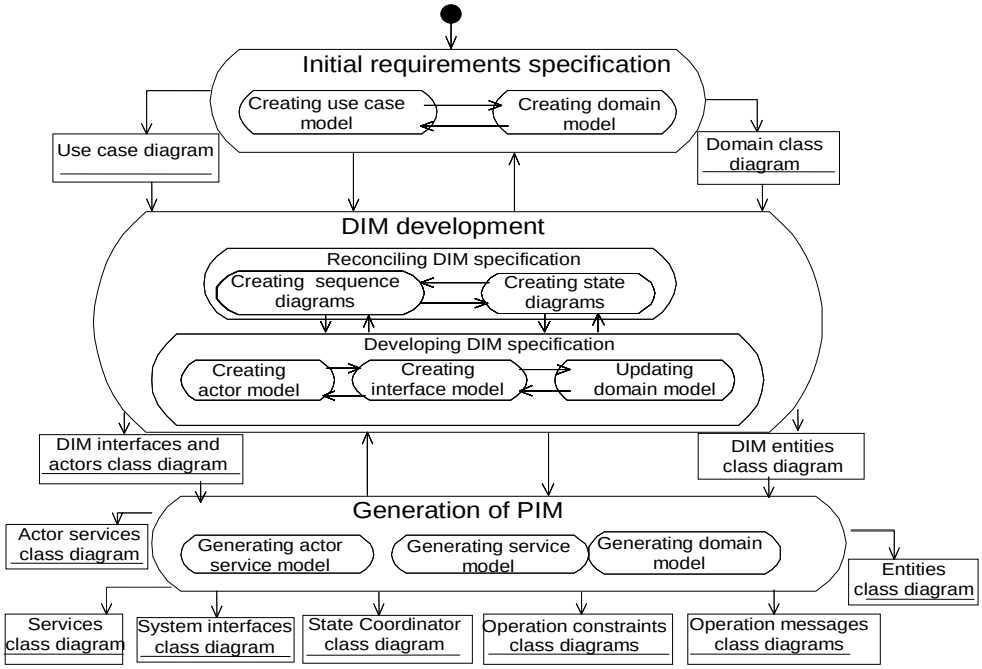

Figure 2. *Method for Requirements Analysis and Transformation to Design*

4. Initial Requirements Definition

Detail requirements model must define state and behaviour of IS. But before creating the formal requirements definition, these requirements are described informally. In this work, the template for initial textual description of requirements is used. This template is based on UML use case diagram and domain model of the system. The domain model serves as a glossary of terms that can be used in writing use cases. Use case model is used to capture the user requirements of the system by detailing all the actions that users can perform. Each action of the use case is described using pre and post conditions. The actors and domain classes in use case description are written starting with capital letter to ensure that they are easy to identify. As new objects are discovered during the use case description the domain model is updated.

5. Design Independent Model

Design Independent Model (DIM) is proposed for definition of requirements for development of wide range of information systems composed of services.

Behaviour of IS is tightly related with its state and has many aspects: interactions, state transitions, control flows and data flows. These aspects are represented by different kinds of UML diagrams comprising views under the same model of target IS. The possibility to achieve consistency of requirements is based on Design Independent Model (DIM). DIM represents overall structure and behaviour of the system, but in contrast to PIM, no design decisions are made in DIM. In DIM, class, sequence and state diagrams are used, supported with OCL constraints; all kinds of diagrams are related. Some diagrams are derivable from others. DIM composition well conforms to requirements for development of services and systems of services where concepts of entities may be distinguished from concepts of services, and interactions are playing the crucial role ensuring "user-oriented" service models.

In general case DIM may be represented visually by stereotyped UML class, sequence, and state diagrams with OCL constraints following principles of precise modelling and contract-based development. In DIM, use cases are mapped to interfaces of the target system, associated with actors – service users – and communicating with external systems (service providers), whose interfaces must be accessed by the target system to provide requested services. Interfaces are defined by the sets of operations identified from steps of use case specifications. Every operation is defined by its signature, pre and post conditions. Besides the interfaces, DIM includes actors, entities and states of entities modelling the state of problem domain.

For development of DIM, one or more sequence diagrams for every use case are constructed from specifications of use cases. These sequence diagrams must represent all desirable interactions between actors (service users), interfaces of the system and possibly the interfaces to external systems, if they are required to fulfil service requests. Sequence diagrams represent interaction related aspect of behavioural requirements. They are suitable for modelling choreography and orchestrations that are of the primary importance for service-oriented systems [Ceponiene, 2005b]. The "engine" of behaviour of object system is the state machine; semantics of functioning information system may be represented by transition system, affected by external events, where interactions between different actors and parts of the system take place and system moves from one compound state to another.

6. Reconciliation of Requirements

Elements of class, sequence diagrams and state machines are interlinked in UML metamodel, but relationships among them have a high degree of abstraction. Such flexibility supports different kinds of usage of UML, but in our case it is necessary to tie models for assurance of consistency between different views of the same system.

The reconciliation of model of system under development is based on integration between states of entities and states of active classes, and integration between interactions and state transitions. The particular attention is devoted to integration of interactions with state machines, which are specialized as state machines of entities and interfaces to behaviour. Explicit separation and reconciliation of states of entities with states of services is also accentuated. Interactions are not directly integrated with state machines in UML metamodel. Such integration is necessary for harmonization of IS model irrespective of development phase. In this work the algorithm is proposed for bidirectional transformations between sequence and state diagrams. There are many works on this topic, but bidirectional transformations are not possible.

For the integration of interactions with state machines UML metamodel was analysed and it was found that for this purpose some adjustments to UML metamodel must be made: in sequence diagram two types of messages are distinguished: Requests and Responses and constraint ensuring connection between Request and Response Messages is created, thus ensuring the possibility of tracking which Response is the answer to which Request (Fig.3); events are specialized into four subtypes: Sent Request, Received Request, Sent Response, and Received Response; information about the senders and receivers of events is captured in state machine, thus ensuring possibility to re-create sequence diagrams from state machines.

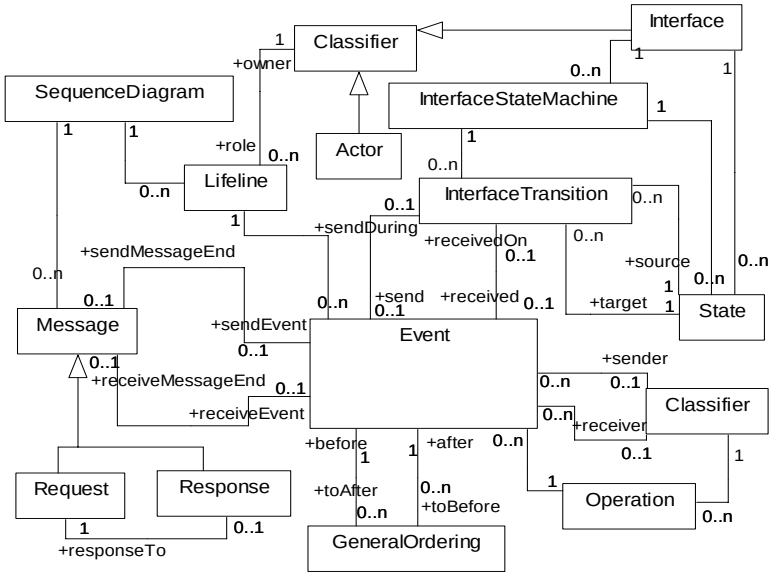

Figure 3. *Fragment of DIM Metamodel Describing Integration of Diagrams*

As events are present both in sequence diagrams and in state machines, the main problem is to map messages, sent between interfaces from sequence diagrams, to states and transitions represented in state machines of these interfaces. The algorithms

for bidirectional transformations between sequence and state diagrams are specified in OCL. The detail description of these algorithms is presented in [Ceponiene et al. (2005c)]. The final definition of requirements is represented by class diagrams of actors, interfaces and entities with respective constraints. It is the most informative view, from which design model may be obtained, but in order to develop this view comprehensively it is necessary to consider and integrate other views.

7. Transformation from Requirements to Design Models

This section is devoted to transformation from design independent model to design (PIM). Design model differs from requirements model in some aspects: design model is supplemented with control classes or components, methods for operations must be elaborated. In this work, the architectural design is considered, during which elements of requirements specification are allocated to architectural elements. For service-oriented design, the State Coordinator pattern was constructed on the base of classical Facade and State patterns. The purpose of Coordinator is the same as of other design patterns: to "normalise" behaviour, discovering recurring activities and concentrating them in separate classes thus making the cohesive units of behaviour. In services execution environment, such recurring behaviour is receiving/sending of messages, checking context and selecting services for execution. Operations of services must be stateless so the information about states is captured by entities, and all constraints describing services subject to state changes are kept in Constraint base. Coordinator and Checker classes perform handling of incoming messages and checking their context; if preconditions of requested services are satisfied (according to information about states of information entities) the acceptance messages are sent and services are delivered according contract, possibly in collaboration with other (internal or external) services. If preconditions are unsatisfied, the exception messages are sent. Checking of postconditions is used for unfolding the expressions of sending messages to other services. The Checker may be implemented in different ways. In a simple case, the pair of checking operations might be created for every service in the system. In an advanced case, it is possible to design the rule engine that is able to add, delete, read, check and transform operation constraints stored declaratively in the base.

The detail description of transformations from DIM to PIM, based on State Coordination pattern, is presented in [Ceponiene et al. (2005b)]. There are three main transformations: DIM actors are transformed to PIM actor services, DIM interfaces are transformed to PIM services, DIM entities are transformed to PIM entities.

8. Implementation of the Proposed Method

For the implementation of the proposed method in UML CASE tools, UML profiles for DIM, PIM and for transformation from DIM to PIM were created. The proposed algorithms were implemented as prototypes in two CASE tools. In ArgoUML tool the

prototype of the module for reconciliation of sequence and state diagrams was implemented, which enables state diagram generation from sequence diagrams and vice versa. Such extensions may be done for various UML CASE tools complying with OMG standards. State Coordinator pattern was implemented using model library in MagicDraw tool where the prototype of plug-in for checking DIM and transforming it to design model was created. The method was used for the development of Publication Agency services system.

The principles applied in this work can be used for a formalization of other suitable design methods and their implementation in CASE tools. Suitable design methods are those that can be described by formal rules. Organizations developing software can use the proposed methodology for automation of their own development methods starting from requirements specification. The proposed methodology would be especially useful for organizations practicing the software development in a large-scale and using the conception of Software Factories.

9. Conclusion

New possibilities to formalize IS design process and make it more „engineering discipline“ can be achieved by separating the requirement definition and design stages, strictly describing requirement and design models and transition between these stages. Such features are not explicitly defined in existing development methods, so design processes remain empiric till now.

For bringing the principles of model driven architecture to the earlier system development stage, comprehensive requirement model must be created which defines the state and behaviour of system independently not only from implementation platform, but also from architectural design. We emphasize “behaviour” because the benefits for design of service-oriented systems and the real potential of Model Driven Architecture are based on rigorous state and behaviour models.

The practical application of the proposed method is more significant for systems which are developed considering their further maintainability. Software service development companies can use the proposed methodology for automation of their own development processes starting from requirements specification. The proposed methodology would be especially useful for the development in large-scale when the support for created systems and reuse of models in similar projects is a challenge.

References

Astesiano, E., Reggio, G. (2002), *Knowledge Structuring and Representation in Requirement Specification,* Technical Report. DISI-Universita di Genova, Italy.

Beck. K. (1999), *Extreme Programming Explained: Embrace Change*, Addison-Wesley, Reading, Massachusetts, ISBN 0201616416.

Ceponiene, L., Nemuraite, L., Paradauskas, B. (2003), *Design of Schemas of State and Behavior for Emerging Information Systems*, in Proc. Pre-Conference Workshop of VLDB 2003, pp. 27-31.

Ceponiene, L., Nemuraite, L. (2005a), *Design independent modeling of information systems using UML and OCL*, Databases and Information Systems: Selected Papers from the Sixth International Baltic Conference DB&IS'2004, Frontiers in Artificial Intelligence and Applications, vol. 118, pp. 224-237.

Ceponiene, L., Nemuraite, L. (2005b), *Transformation from Requirements to Design for Service Oriented Information Systems*, in Proc. ADBIS 2005: Advances in Databases and Information Systems, Tallinn, Estonia, pp. 164-177.

Ceponiene, L., Nemuraite, L. (2005c), *Transformations of UML diagrams for reconciliation of requirements*, in Proc. 13th International Conference Information Systems Development: Advances in Theory, Practice, and Education, vol. 28, pp. 289-301.

D'Souza, D. F., Wills, A. C. (1999), *Objects, Components, and Frameworks with UML. The Catalysis Approach*, Addison Wesley, ISBN 0201310120.

Dedene, G., Snoeck, M. (1994), *M.E.R.O.DE.: A Model-driven Entity-Relationship Object-oriented DEvelopment method*, ACM SIGSOFT Software Engineering Notes, vol. 13, no. 3, pp.51-61.

Frankel, D. (2003), *Model Driven Architecture: Applying MDA to Enterprise Computing*, John Wiley & Sons, ISBN 0-471-31920-1.

Gamma, E., Helm, R., Johnson, R., Vlissides, J. (1995), *Design Patterns: Elements of Reusable Object-Oriented Software*, Addison-Wesley Professional, ISBN 0201633612.

Jacobson, I., Booch, G., Rumbaugh, J. (1999), *The Unified Software Development process*, Addison-Wesley Longman Publishing Co., ISBN 0201571692.

Mäkinen, E., Systä, T. (2000), *An Interactive approach for synthesizing UML statechart diagrams from Sequence Diagrams*, in Proc. OOPSLA 2000 Workshop Scenario-based round-trip engineering, pp. 7-12.

OMG (2003a), *Unified Modeling Language Superstructure Specification. Version 2.0*, OMG Document ptc/03-08-02

OMG (2003b), *Unified Modeling Language: OCL Version 2.0*, OMG Document ptc/03-08-08.

OMG (2006), *UML Profile and Metamodel for Services (UPMS), Request For Proposal*, OMG Document soa/06-09-09.

Reggio, G., Cerioli, M., Astesiano, E. (2001), *Towards a Rigorous Semantics of UML Supporting Its Multiview Approach*, LNCS, vol. 2029, pp. 171 - 186.

Rosenberg, D., Scott, K. (2001), *Applying Use Case Driven Object Modeling with UML: An Annotated e-Commerce Example*, Addison Wesley, ISBN 0201730391.

Whittle, J., Schumann, J. (2000), *Generating Statechart Designs From Scenarios*, in Proc. 22nd International Conference on Software Engineering, pp. 314-323.