AUTOMATED TRANSFORMATION OF USE CASE MAPS TO UML ACTIVITY DIAGRAMS

Yasser A. Khan and Mohamed El-Attar
Department of Information and Computer Science
King Fahd University of Petroleum and Minerals
Dhahran, Kingdom of Saudi Arabia
{yasera, melattar}@kfupm.edu.sa

Keywords: Use Case Map, UML, Activity Diagram, Model Transformation, ATL

Abstract: Use Case Maps (UCMs) is a modeling language that has been successfully deployed to model the behaviour of many types of systems, such as telecommunication systems, web applications, agent based systems and operating systems. However, as a high-level modeling language there exist a conceptual gap between UCMs and code-level. The crucial responsibility of filling this gap is usually undertaken by programmers, which leaves them prone to development mistakes and delivering an end system that does not accurately realize the behaviour specified in UCMs. UML activity diagrams (ADs) share many concepts with UCMs. Unlike UCMs however, many approaches have been proposed and implemented that transform ADs into other design artefacts. In particular, ADs can be transformed into class diagrams which can be used to generate code skeleton and ensure consistency between the programming and modeling efforts. To capitalize on the advantages of ADs, a model transformation approach from UCM notation to AD notation has been implemented and proposed in this paper. A case study is presented to illustrate the application of the proposed model transformation approach. The generated AD was thoroughly inspected and verified by the authors in addition to three independent Software Engineering professors at the host institution.

1 INTRODUCTION

The Use Case Map (UCM) notation is a visual modeling language that allows the high-level description of object-oriented systems. It was first introduced by Buhr and Casselman in the mid-1990s. Over the years UCM notation has gained attention from both researchers and industry. It has been successfully used for telecommunication systems (Amyot and Logrippo, 2000; Andrade, 2000), web applications (Amyot, Roy and Weiss, 2005), agent based systems (Amyot, et al., 2004; Elammari and Lalonde, 1999) and operating systems (Billard, 2004).

While the benefits of UCMs have been evident in industry and reported in the literature, there remains a conceptual gap between UCMs and code-level which is required by programmers to fill. Consequently, programmers may develop a system that does not accurately represent the behaviour modelled in the UCM. This problem can be averted in object-oriented systems if UCMs can be seamlessly mapped to UML class diagrams, which is the de-facto diagram used to model the structural characteristics of object-oriented systems. A key advantage of class diagrams is that they can be transformed seamlessly into code-skeleton which in turn greatly aids the alignment of programming and modeling efforts. However, to date, UCM remains not part of the UML modeling language. As such, there lacks research in the area of transforming UCMs into UML design models, such as UML activity, sequence, statechart and class diagrams. On the other hand, a very large number of transformation solutions have been proposed in the literature to transform UML diagrams into other UML diagrams. Some transformations have been proposed for downstream development, such as from activity to class diagrams. Other transformations have been proposed to aid reverse engineering efforts, such as from class to sequence diagrams.

This paper proposes a model transformation approach to transform a given UCM into a UML 2.2 Activity Diagram (AD). The model transformation approach will systematically produce a consistent and accurate representation of UCMs in the form of ADs. Defining a formal model transformation approach has the obvious advantage of avoiding human errors which would otherwise be injected if the transformation was performed manually. In object-oriented software development projects, the generated ADs will greatly ensure that the developed end system accurately represents the behaviour modelled originally in the UCM diagrams.

The remainder of this paper is organized as follows. Section 2 gives a brief introduction to UCM and AD notation. In Section 3, mappings between UCM and AD notation are presented. Section 4 describes a case study to illustrate the mapping. Section 5 briefly discusses related works and, Section 6 concludes and discusses future work.

2 BACKGROUND

2.1 UCM Notation

A UCM consists of one or more paths each of which represent a use case scenario. A path starts at a *start point* (filled circle) and ends at an *end point* (bar). The actions performed by the system or use case actor along these paths are *responsibilities* (cross). These responsibilities can be bound to *components—actors, agents, teams, objects and processes.* An *actor* component (rectangle including a stickman) represents a stakeholder who is associated with the system through a number of usage scenarios. Software agents in agent-oriented systems can be represented by the *agent* component (rectangle with a dark border). *Teams* (rectangle) represent high level abstract components that can be further decomposed into multiple levels of other component types. However, *objects* (box with rounded corners), which represent instances of a class, cannot be further decomposed. *Processes* (slanted rectangle) are executing components of a system and may include *object* components. An *OR-fork* divides a path into one or more alternative paths based on a guard condition. Concurrent paths emerge from *AND-forks* (bar). Common paths are merged by *OR-joins* and concurrent paths are synchronized by *AND-joins* (bar). *Stubs* (diamond) are containers for nested maps. Stubs are useful for refactoring complex UCMs via modularization. Erroneous situations that may stop the flow of a path are represented by *failure points* (ground). *Timers* (clock) express the amount of time to wait before a path can progress further. A *waiting place* (filled circle and bar) allows a path to wait for another path to finish before it can continue. The interested reader may refer to Buhr and Casselman's (1996) book on UCMs for more details on its notation.

2.2 UML 2.2 AD Notation

An *activity* in an AD is a directed graph comprising of *activity nodes* and *activity edges*. The Object Management Group's (OMG) UML superstructure specification defines three types of activity nodes— *action nodes*, *object nodes* and *control nodes* (OMG, 2009). *Control flow* is an activity edge that represents the transitions between activity nodes. Action nodes exchange messages with each other through the *object flow* edge. Both control and object flows are represented as an arrow. Action nodes (box with rounded corners) represent the actions to be performed by the system being modelled within a particular context. The exchange of messages between actions is modelled by *object nodes.* Control nodes coordinate the execution of an AD. The flow of an activity starts at an *initial node* (solid circle) and stops at a *final node* (solid circle surrounded by hollow circle). Concurrent flows of control emerge from *fork nodes*. Alternate flows of control initiate from *decision nodes*. *Join nodes* synchronize concurrent flows and *merge nodes* combine alternate flows. *Activity partitions* or *swimlanes* are regions on an activity surrounded by parallel lines, either horizontal or vertical. They group related nodes together, represent organizational units such as classes (Booch, Rumbaugh and Jacobson, 2005) and may nest other partitions. A *structured activity node* (dashed box with rounded corners) is defined as "an executable activity node that may have an expansion into subordinate nodes as an ActivityGroup" (OMG, 2009). *ActivityGroup* refers to an abstract metaclass in the UML 2.2 metamodel that groups set of nodes and edges of an activity (Weilkiens and Oestereich, 2006). Activity partitions and structured activity nodes inherit from this metaclass. The interested reader may refer to the OMG UML 2.2 specification for more details on AD notation (OMG, 2009).

3 UCM TO AD MAPPING

UCMs and ADs share similar concepts. The definitions of UCM constructs given by Buhr and

Casselman (1996) were used. For AD constructs, the definitions provided in the OMG UML 2.2 specification (OMG, 2009) were used. The definitions obtained for UCM and AD constructs were used to propose mappings between the UCM and AD notations.

A UCM is composed of one or more paths. Each path describes a particular scenario. An activity in an AD can also contain multiple flows of control. Hence, mappings between UCMs and ADs are proposed. Start points which represent the initiation of a UCM path are mapped to UML initial nodes. UCM end points which represent the termination of UCM path are mapped to UML final nodes.

The OMG UML 2.2 specification defines an opaque action as "an action with implementation-specific semantics". Since UCM responsibilities are high level descriptions of system behaviour, they are mapped to opaque actions. Buhr and Casselman (1996) define a timer as "a special kind of responsibility along a path that takes up real time without taking up processing resources". Based on this analogy timers are mapped to opaque actions as well, similar to the mapping of responsibilities except that a 'No Action' label is appended to the timer's notation to distinguish it from other opaque actions. UCM failure points are defined as "points where a path may end abnormally, due to some failure in the underlying system". They simply indicate the possible occurrence of a failure or exception and thus they are mapped to opaque actions. A 'Handle Exception' is appended to the failure point's notation to distinguish it from other opaque actions.

UCM concurrency and branching constructs are intuitively mapped to their respective AD constructs. It should be noted that concurrent control flows in ADs are required to synchronize at a join node; however, UCMs have no such restriction (Amyot and Mussbacher, 2000).

The UCM elements which are bound to components (teams, objects, processes, actors, and agents) are grouped into activity partitions. This mapping decision is made since their purpose is to group related activity nodes together and to represent organizational units such as classes (Booch, Rumbaugh and Jacobson, 2005). The difference between these notations is that UCM components cannot share elements (responsibilities, timers, failure points, etc…) whereas ADs have no such restriction. ADs allow activity partitions to overlap, enabling them to share nodes and edges. Hierarchical decomposition of activity partitions in ADs is similar to that of components in UCMs. In

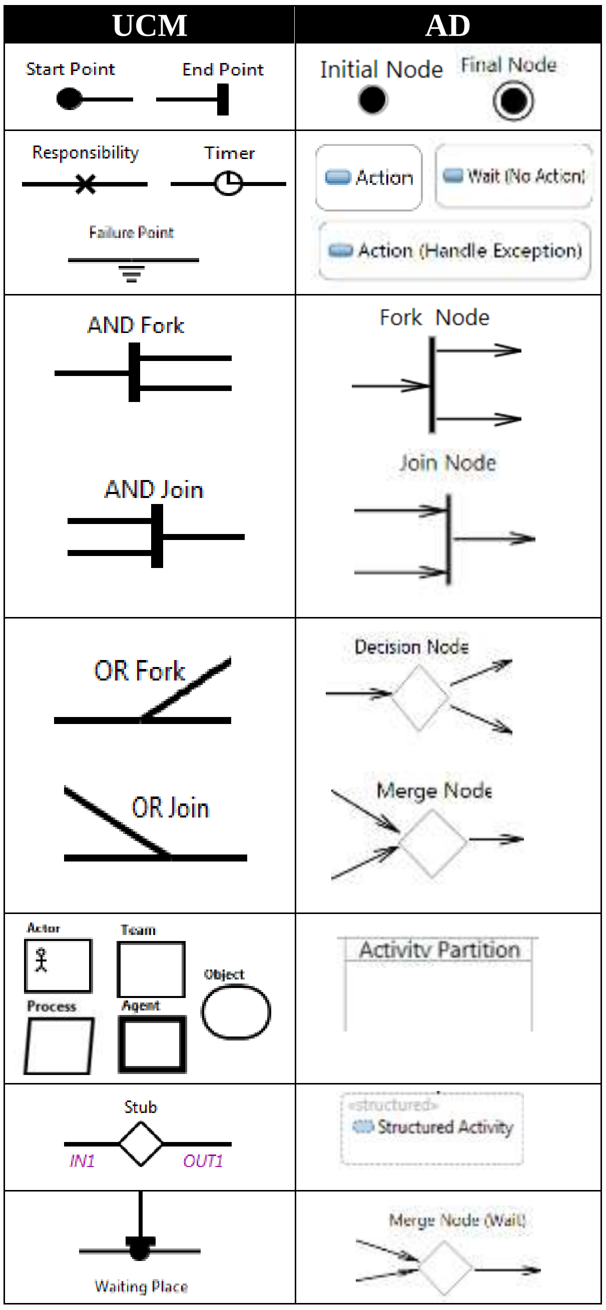

Figure 1: Mapping of UCM to AD notation

order to determine which type of component (actor, process, object, etc....) they correspond to, we suggest their names be appended with the type of component they correspond to. Names of activity partitions that correspond to generic components (of no specific type) are appended with an '(Other)' label.

Stubs which represent nested UCMs are mapped to structured activities, which cannot share nodes and edges with other structured activities. This mapping decision is made because stubs are individual UCMs by themselves which do not share elements (responsibilities, timers, failure points, etc...) with parent or child maps. It should be noted that components inside a stub will be ignored by our mapping, since structured activities cannot include activity partitions. However, nesting of structured activities is allowed as is the case with stubs in UCMs. In order to prevent loss of information while using this mapping, UCM designers should model stubs such that they are contained within a component.

UCM waiting points are points along a path that indicate that execution flow must wait for events along another path (Buhr and Casselman, 1996). There is no such notation in ADs that can allow a control flow to wait for another one. We propose to use merge nodes with labels appended by 'Wait', to depict such behaviour in a flow. It should be noted that the end point that is connected to a waiting point is discarded during the mapping. Otherwise, it would be mapped to an activity final node, which would be connected to a merge node (waiting). This mapping decision is made since a final node stops a flow in an activity. A visual summary of the mappings is shown in Figure 1.

4 CASE STUDY

In this section a case study is presented to illustrate the proposed mapping. The mapping was implemented using ATL, a model transformation language. ATL transformation is applied to the UCM (Figure 2) of an Elevator Control System (ECS), which is available at (Amyot, 2001). It UCM pertains to an elevator control system which was adapted from *"Designing Concurrent, Distributed and Real-Time Applications with UML"* (Gomaa, 2000). It represents the functionality of an ECS that controls one or more elevators. The two main responsibilities of the system are to respond to elevator calls from users and to manage the motion of the elevators between floors.

A use case begins with a request from the user to call the elevator to go to above or below levels. The request gets queued with other call requests. Depending on the state of the elevator whether it is stationary or moving, the system will control motor actions to move the elevator appropriately. Once the elevator approaches a requested floor the motor stops, the door opens and the corresponding call request is removed from the queue.

This model was selected since it includes most of the UCM notational set and represents a complex scenario with multiple alternates. The source model (Figure 2) was provided as input to the ATL transformation algorithm defined, which resulted in the generation of the AD shown in Figure 3. The ATL source code, source and target models are available to the interested reader for download at (Khan, 2011).

The target model was thoroughly inspected and verified by three Software Engineering professors at the host institution. The proposed mapping (Figure 1) was given to them along with the source and target models. Reviewers indicated confusion while distinguishing between decision nodes and merge nodes. This confusion is due to the fact that the Eclipse UML 2 tools do not include separate notation for decision nodes. Merge nodes are intended to be used in place of decision nodes. Hence, to avoid this confusion labels are placed on their respective notations. Another reviewer indicated confusion while interpreting edges coming in and out of fork and join nodes. This was found to be a layout issue. The transformation results in the model elements being placed in a default layout. The target model was manually realigned to clear the confusion. The same reviewer indicated that the proposed mapping did not consider dynamic stubs. Hence, a mapping for dynamic stubs was implemented in ATL. This can be found in the available source code. We plan to validate this mapping on a case study as part of our future work.

5 RELATED WORK

Amyot and Mussbacher (2000) proposed an extension of UML 1.3 with UCM core concepts for the purpose of introducing a new "UCM View" to the existing set of UML views. To date, the proposed "UCM View" is not a UML standard. Hence, there is a need for a mapping between these distinct notations. Bordeleau and Buhr (1997) proposed modeling steps from UCMs to ROOM state machines (Selic, Gullekson and Ward, 1994). These steps help in bridging the conceptual gap between the notations and enables traceability from detailed design to scenarios. A method for deriving SDL diagrams (Ellsberger, Hogrefe & Sarma, 1997) from UCMs is given by Sales and Probert (1996). This study has also not only proposed mappings between modeling notations (UCM and UML AD) but also suggests automation of mappings. Automation will not only enable traceability but also

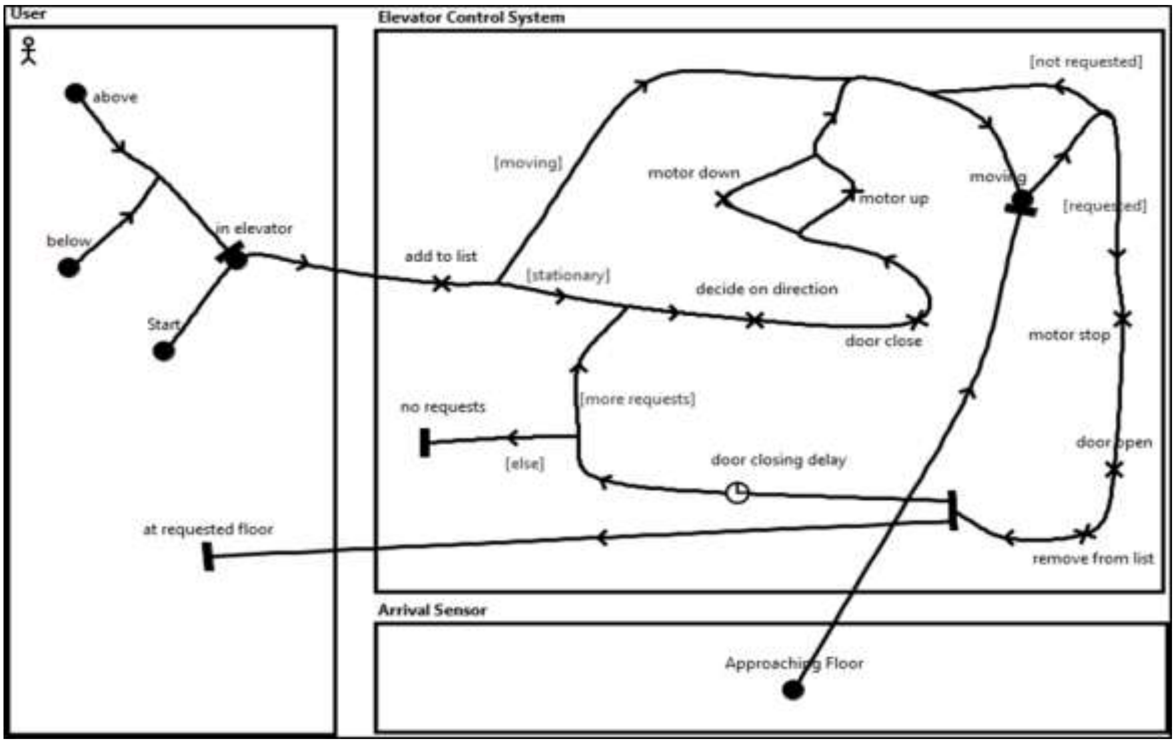

Figure 2: Elevator Control System source model

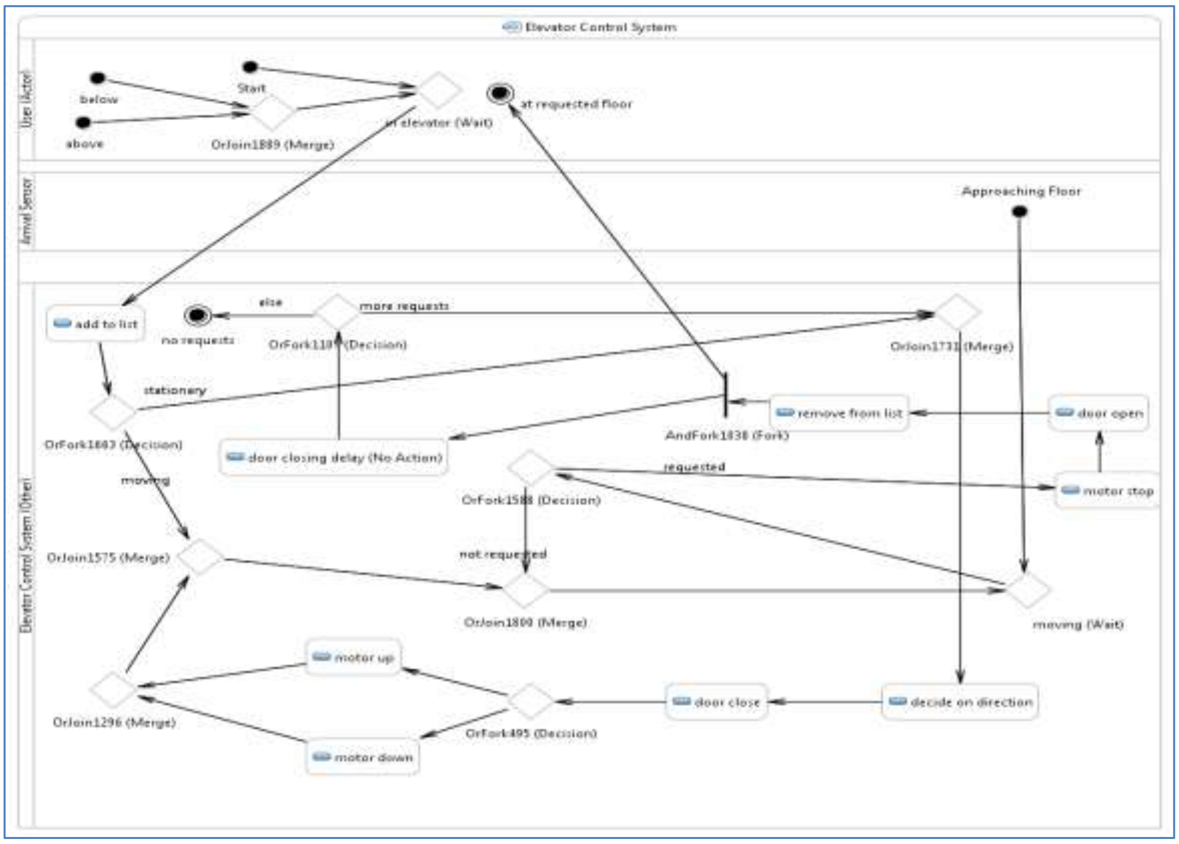

Figure 3: Elevator Control System target model

reduce the effort required to derive detailed design from scenarios represented as UCMs. He, Amyot and Williams (2003) illustrated the generation of Message Sequence Charts (ITU-T, 2001) from UCMs using an automated tool. To the best of our knowledge, no such tool exists for automating the conversion of UCMs to UML ADs. The mappings and transformations presented in this study can be used for the development of such a tool. Moreover, no attempt has been made to propose mappings between UCM and UML 2.x notation

6 CONCLUSION

Usage of this transformation can enable consistent communication between requirements engineers and designers/developers involved in a software development project. The requirements engineers can model use case scenarios using UCMs. The designers/developers who are not familiar with the UCM notation can use this transformation to convert them to ADs, which are part of UML, the de-facto standard for documenting design. This paper contributes to the Model Driven Engineering (MDE) software development methodology, which relies on automated transformation of design documents.

The target models (ADs) produced by the transformation are specific to the Eclipse UML 2 tools. Tools, such as Enterprise Architect and Rational Rose allow designers to import/export platform independent models. Our, future work involves implementing this mapping to produce platform independent ADs, which can be imported into other platforms. Mappings to sequence and statechart diagrams are also part of our future work.

REFERENCES

Amyot, D. & Mussbacher, G. 2000. On the extension of UML with use case maps concepts. *Proceedings of the 3rd international conference on The unified modeling language: advancing the standard.* York, UK: Springer-Verlag.

Amyot, D. 2001. Bridging the gap between requirements and design with use case maps. [online] Available at: <http://people.scs.carleton.ca/~jeanpier/304/Amyot.PDF> [Accessed: 15 October 2011]

Amyot, D., Amyot, D., Amyot, D., Amyot, D., Abdelaziz, T., Elammari, M. & Unland, R. 2004. Visualizing a Multiagent-Based Medical Diagnosis System Using a Methodology Based on Use Case Maps. *MATES'04.*

Amyot, D., Roy, J-F. & Weiss, M. 2005. UCM-Driven testing of web applications. *Proceedings of the 12th international conference on Model Driven.* Grimstad, Norway: Springer-Verlag.

Andrade, R. M. C. 2000. Applying Use Case Maps and Formal Methods to the Development of Wireless Mobile ATM Networks. *the Proc. of the Fifth NASA Langley Formal Methods Workshop.*

Billard, E. A. 2004. Operating system scenarios as Use Case Maps. *Proceedings of the 4th international workshop on Software and performance.* Redwood Shores, California: ACM.

Bordeleau, F. & Buhr, R. J. A. 1997. UCM-ROOM modeling: from use case maps to communicating state machines. *Proceedings of the 1997 international conference on Engineering of computer-based systems.* Monterey, California: IEEE Computer Society.

Buhr, R. J. A. & Casselman, R. S. 1996. *Use case maps for object-oriented systems*, Prentice Hall.

Elammari, M. & Lalonde, W. 1999. An Agent-Oriented Methodology: High-Level and Intermediate Models. *Proc. of the 1st Int. Workshop. on Agent-Oriented Information Systems.*

Ellsberger, J., Hogrefe, D. & Sarma, A. 1997. *SDL: formal object-oriented language for communicating systems*, Prentice Hall.

Gomaa, H. 2000. *Designing concurrent, distributed, and real-time applications with UML*, Addison-Wesley.

He, Y., Amyot, D. & Williams, A. W. 2003. Synthesizing SDL from use case maps: an experiment. *Proceedings of the 11th international conference on System design.* Stuttgart, Germany: Springer-Verlag.

ITU-T: Recommendation Z. 120, 2001. Message Sequence Chart (MSC). Geneva, Switzerland.

Khan, Y. Transforming UCMs to ADs - the ATL Code. Available at: <http ://sourceforge.net/projects/ucmtoumlad>. [Accessed: 19 March 2012]

Object Management Group (OMG), 2009. *OMG Unified Modeling Language (OMG UML) Superstructure*. [online] Available at: <http://www.omg.org/spec/UML/2.2/Superstructure/PDF> [Accessed: 19 October 2011]

Rumbaugh, J., Jacobson, I. & Booch, G. 2005. *The unified modeling language reference manual*, Addison-Wesley.

Sales, I. S. & Probert, R. L. 2000. From High-Level Behaviour to High-Level Design: Use Case Maps to Specification and Description Language. *In: SBRC'2000, 18th Brazilian Symposium on Computer Networks.*

Selic, B., Gullekson, G. & Ward, P. T. 1994. *Real-time object-oriented modeling*, Wiley & Sons.

Weilkiens, T. & Oestereich, B. 2006. *UML 2 Certification Guide: Fundamental and Intermediate Exams*, Elsevier/Morgan Kaufmann.