

UML-Intensive Framework for Modeling Software Requirements

Dr. Darius Silingas, Prof. Rimantas Butleris

Department of Information Systems, Kaunas University of Technology

Darius.Silingas@bpi.lt, Rimantas.Butleris@ktu.lt

Abstract. Investigation of software projects has shown that requirements analysis is one of the most problematic activities in software development. Textual requirements specifications are difficult to develop, understand, review, and maintain. Graphical modeling is widely recognized as a more effective analysis tool. Software industry has adopted UML (Unified Modeling Language) as *de facto* standard in software modeling. UML defines a powerful, but also difficult to learn, modeling toolkit: 13 types of diagrams, more than 100 inter-related metaclasses used as modeling concepts, and possibility to define custom extensions. Since UML doesn't define modeling method, practitioners lack guidance on how to apply it efficiently to modeling software requirements, and apply it only fragmentally losing many benefits that UML provides. In this paper, we present the analysis of modern requirements modeling techniques. Based on analysis results, we discuss how various domain and requirements analysis elements – semantic map of business concepts, lifecycles of business objects, business processes, business rules, system context diagram, use cases and their scenarios, constraints, and user interface prototypes – can be modeled using UML. We propose UML extensions and a practical UML-intensive framework necessary for concise requirements modeling. The application of this framework is demonstrated by modeling a case study – software system for library management – using customized MagicDraw UML environment. Our work is important for practitioners trying to adopt UML for requirements analysis and for scientists working on creating more detailed requirements analysis methods based on UML.

1. Current Situation in Requirements Analysis

Software development is getting more mature by advancing development processes, methods, and tools. However, the famous *CHAOS Report* statistics, published by Standish Group show that still only about one third of software projects can be called successful, i.e. they reach their goals within planned budget and on time [1]. Research on *post-mortem* projects' analysis shows that the major problems come from requirements elicitation, analysis, specification, and management. Deploying successful requirements process in a concrete organization is an important issue [2, 3]. While companies continue to use text-based documents as major means for specifying and analyzing requirements, the graphical requirements modeling are getting increasingly more attention in industry. This trend increased after *Object Management Group* (OMG) standardized *Unified Modeling Language* (UML) [4], which has become *de facto* standard in industry. A well-known saying tells us that *one picture is worth a thousand words*. It also applies in requirements analysis, where business people have to communicate with software developers, who do not know their domain and speak a different – technical – language. Additionally, UML tools support refining requirements models with design and implementation details for enabling traceability, validation, prototyping, code generation and other benefits. In large software development projects, these features are very important for evolving and managing requirement models. However, there are some practical problems with UML complexity and lack of unified method or framework for requirements engineering [5]. Practitioners and scientists propose different approaches for eliciting and analyzing software requirements. The most popular method used in modern requirements analysis is *use cases*. It was invented in Ericsson, popularized by Ivar Jacobson [6] and adopted by numerous companies, and described in requirements engineering textbooks [7]. UML provides *Use Case* diagram for visualizing use case analysis artifacts. However, requirements analysis is not limited to use cases. In fact, they capture only end user-level functional requirements. A lot of research is also made in specifying business goals and processes, performing domain analysis. Although it was shown that UML might be extended and used for business modeling, the business modelers' community was not satisfied by UML, and created a new *Business Process Modeling Notation* (BPMN) [8], which has become OMG standard as well. In many cases, they also apply *IDEF* notations [9]. In domain analysis, analysts continue to apply old-style *Entity Relationship* (ER) notation, which was popular in database design since 70s [10]. A lot of attention is paid to business goals [11], business rules, business object lifecycles, business roles and processes in organization, which also can be done using UML [12]. Real-time and embedded system developers have also come up with a different flavor of UML – *System Modeling Language* (SysML) [13], which defines *Requirements* diagram and enables capturing various non-functional and detailed functional requirements and defining specific links between requirements and other model elements. Most popular requirements textbooks [7, 14] introduce various diagrams based on both UML and other informal notations, e.g. system context diagram, and hand-drawn user interface prototypes. All of the mentioned requirements artifacts can be modeled using UML. Since UML is a general-purpose modeling language with more than 100 modeling elements (UML metaclasses) and without standardized method,

practitioners apply it only fragmentally, and do not make use of its powerful capabilities to define consistent, integrated, and reusable requirements models. In [23] one of the authors of this paper discussed the framework for creating UML models for MDD (Model-Driven Development). This paper extends it with more focus on the details of a specific part of this framework – applying UML for requirements modeling.

2. Applying UML for Requirements Modeling

We will provide some guidance on how to use UML diagrams for different requirements analysis purposes and how to map conventional requirements artifacts to UML elements. We will also present and propose some UML extensions or modeling aspects that are necessary for ensuring that various requirements elements that are mapped to the same UML element can be differentiated. Because of paper size limitation, we do not discuss pros and cons of the proposed mapping – here we just show that reasonable mapping is possible.

Table 1. Mapping some of the most popular requirements artifacts to UML metaclasses

Requirements artifact	UML metaclass	Stereotype	Recommendations
Business concept	Class	-	Attributes and operations should be hidden in diagrams.
Business concept relationship	Association	-	Typically modelers specify association name and role multiplicities or roles names, multiplicities, and navigability.
Business object lifecycle	State Machine	-	Should be nested within business concept.
Business goal	Use Case	Goal	All of these elements should be placed in a specific <i>Package</i> or <i>Model</i> element, which is dedicated for business modeling.
Business role	Actor	Role	
Business process	Activity	Process	
Business task	Action	Task	
Business rule	Constraint, Guard	-	Might be specified formally using OCL or informally with simple text.
Business fact	Instance Specification, Comment	-	Data facts can be defined using instance specifications in object diagram. Other types of business facts can be captured using simple comments.
System	Component	-	The component label may be hidden.
Document (information) form	Class	Document	Implementation specific information such as visibility, derivation should be hidden.
Document (information) sample	Instance Specification	Sample	Document should be created
Information flow	Information Flow	-	Information flow should be mapped on associations between classifiers, e.g. classes, components, activities
User group	Actor	-	Can be grouped into primary, secondary, system, and pseudo.
User task	Use Case	-	Extensions might be used for documentation.
Usage scenario	Activity, Interaction	-	Should be placed inside use case. Once use case may have multiple scenarios.
Non-functional requirement	Class	Requirement	Stereotype tags or stereotypes should be used for specifying different categories of requirements.
Time constraint	Duration Constraint	-	Can be visualized in sequence or timing diagrams
GUI navigation schema	State Machine	-	State represents working in a specific screen, transition – navigation between screens, trigger – GUI event.
GUI prototype	Structured Class	-	Composite structure diagram can be used for abstract prototyping.
Refinement	Abstraction	-	Can be used for generating relationship matrices
Composition	-	-	Can be modeled by nesting requirements.

You can also find critics on using UML as requirements specification language [15], but we believe most of the issues can be solved using appropriate UML tool with rich possibilities for modeling environment customization and extensions, e.g. MagicDraw UML. On the other hand, there are also suggestions to use more UML for requirements analysis and visualizations [16]. Multiple authors provide numerous papers on more detailed approaches to customizing UML for specific requirements modeling needs, e.g. analyzing scenarios [17], modeling user interface prototypes [18], refining requirements [19], etc. Some researchers also suggest that UML can be specialized for early phase requirements gathering process [20], but we believe that early phase modeling should be focusing on the same types of artifacts with less details (details should be refined later).

3. Framework for UML-Based Requirements Modeling with Demonstrated Samples

Due to the fact that practitioners apply different means for modeling various requirements artifacts, it is not typical to create concise requirements models that include and relate various types of requirements artifacts. The lack of framework for guiding requirements models is one of the main issues. In academic community, researchers propose many detailed and focused requirements development methods [21, 22]. However, most of these methods resulting from academic research are too complex for practical application and solve just specific specialized issues. A simple and adaptable framework for requirements modeling with demonstrated samples created using available tools on a realistic case study gives much more value for practitioners. We propose to use requirements modeling part from the UML-intensive framework for model-driven development described in [23]. We will give short descriptions and samples for each of the requirements modeling artifacts.

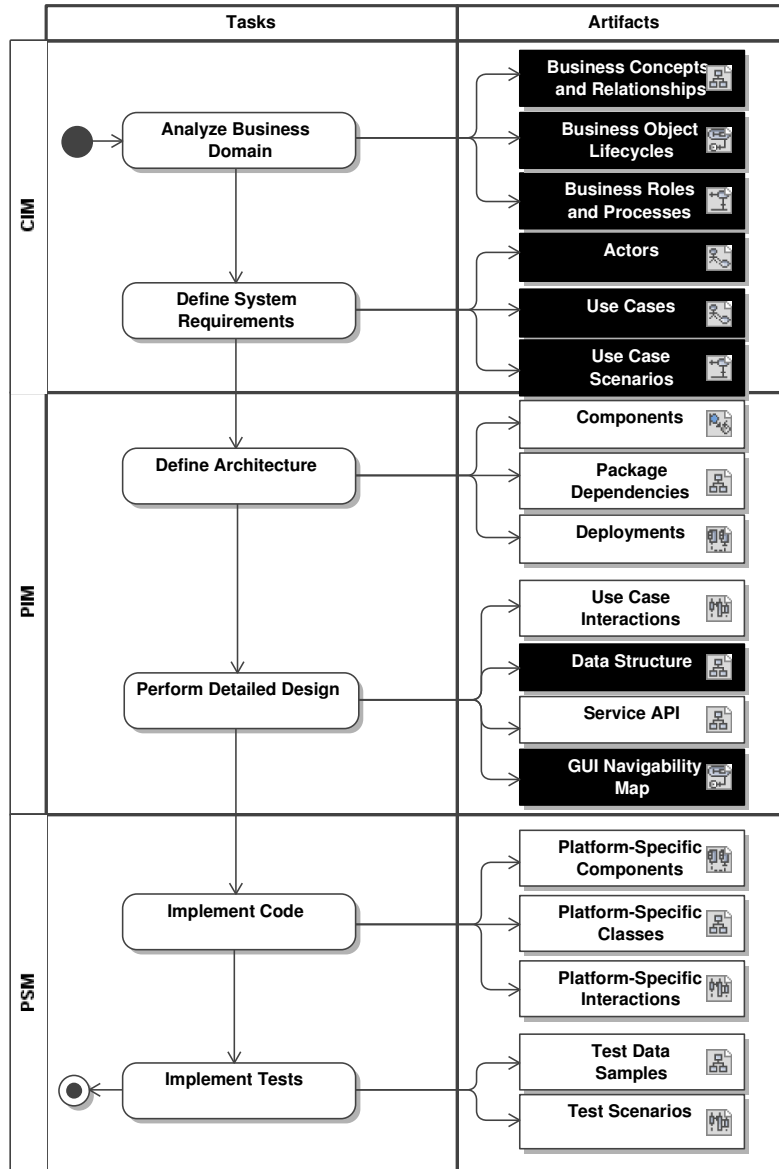


Figure 1. UML-intensive framework for model-driven development with emphasized requirements modeling artifacts.

Different methodologists disagree on how to start development of business information systems. We suggest that the starting point should be business concept identification and analysis of their relationships. For this purpose we can apply simple class diagram using only classes with names and without more detailed information, associations with names and role multiplicities. Such class diagrams are discussed by business analysts and domain experts who are usually not familiar with object-oriented analysis and design (OOAD) and UML notation. Therefore, it is very important that all the other elements of class diagrams, such as aggregations, compositions, generalizations, interfaces, enumerations, etc., should not be used for conceptual analysis. Keeping it simple enables even OOAD/UML novices to understanding it after getting a little explanation. Additionally, you can provide textual descriptions for each of these concepts and generate printable or navigable domain vocabularies. We believe this should be the first artifact since it sets up the vocabulary that should be used for defining other requirement model elements (business processes, tasks, use cases, etc.).

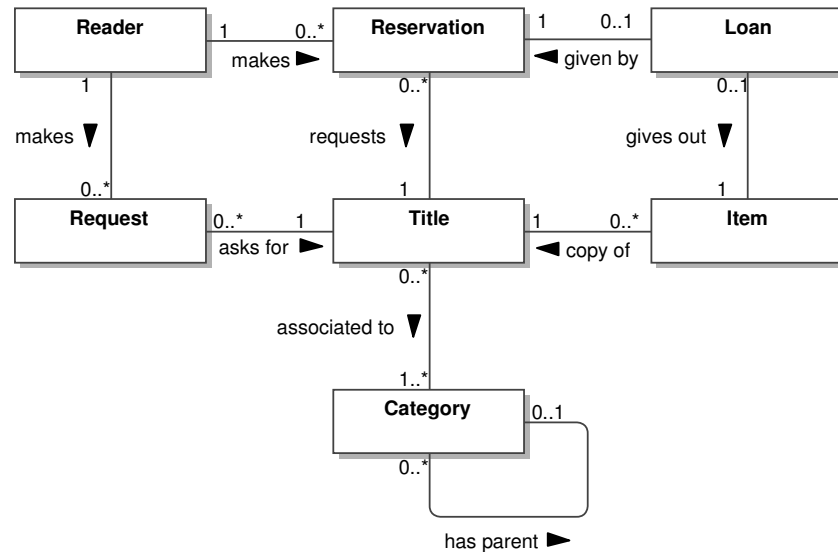


Figure 2. UML class diagram visualizing business concepts and relationships.

Organizations have business rules for managing business objects. In many cases, business rules regulate how important business objects change states and are applicable only when object is in a particular state. For analyzing how business object change their states during their lifecycle you should use state machine diagrams. The states also serve as a part of terminology, which will be used in other business and requirements models. State machine diagrams should be created only for those business concepts that have dynamic states. Business modelers should define triggers on all transitions in state diagram. In business modeling for transition triggers most people use informal *signals* that in most cases correspond to actions of business roles. Also, *time* and *property change* triggers are used to express various states changes according to time- or data-based business rules. It is possible to define inner triggers that happen inside one state and doesn't fire a transition.

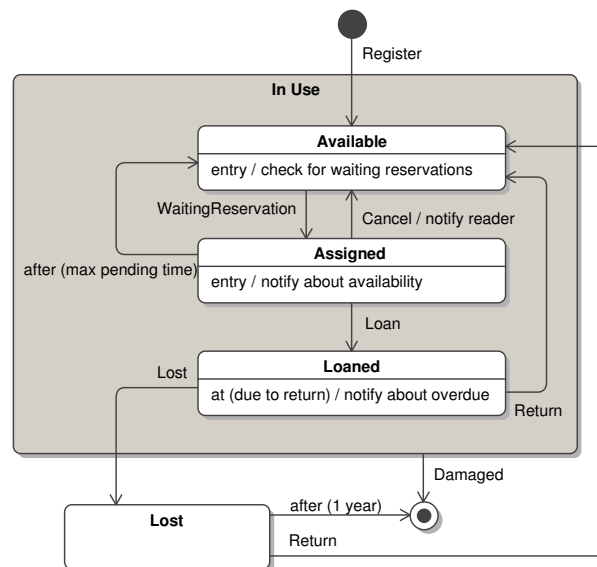


Figure 3. UML state machine diagram visualizing *Item* business object lifecycle.

After learning domain terminology and business rules concerning lifecycles of business objects, we can identify business roles and business processes, and associate roles to processes in which they are involved. We recommend to model business roles with actors, and business processes as UML activity or BPMN diagram. If modelers need to visually separate it from system actors and use cases. The business roles association to business processes is best done within specialized use case diagram or editable relationship matrix. In the picture below we present BPMN diagram, showing top-level business processes from *Reader* and *Librarian* role perspectives. In BPMN diagram it is typical to model lanes for representing different perspectives or different companies. The major elements inside one lane are tasks or sub-processes that enable drill-in functionality for creating and navigating to more detailed business process diagrams. Modelers also use transitions, different types of events and gateways, and data objects that may represent instances of previously identified business concepts.

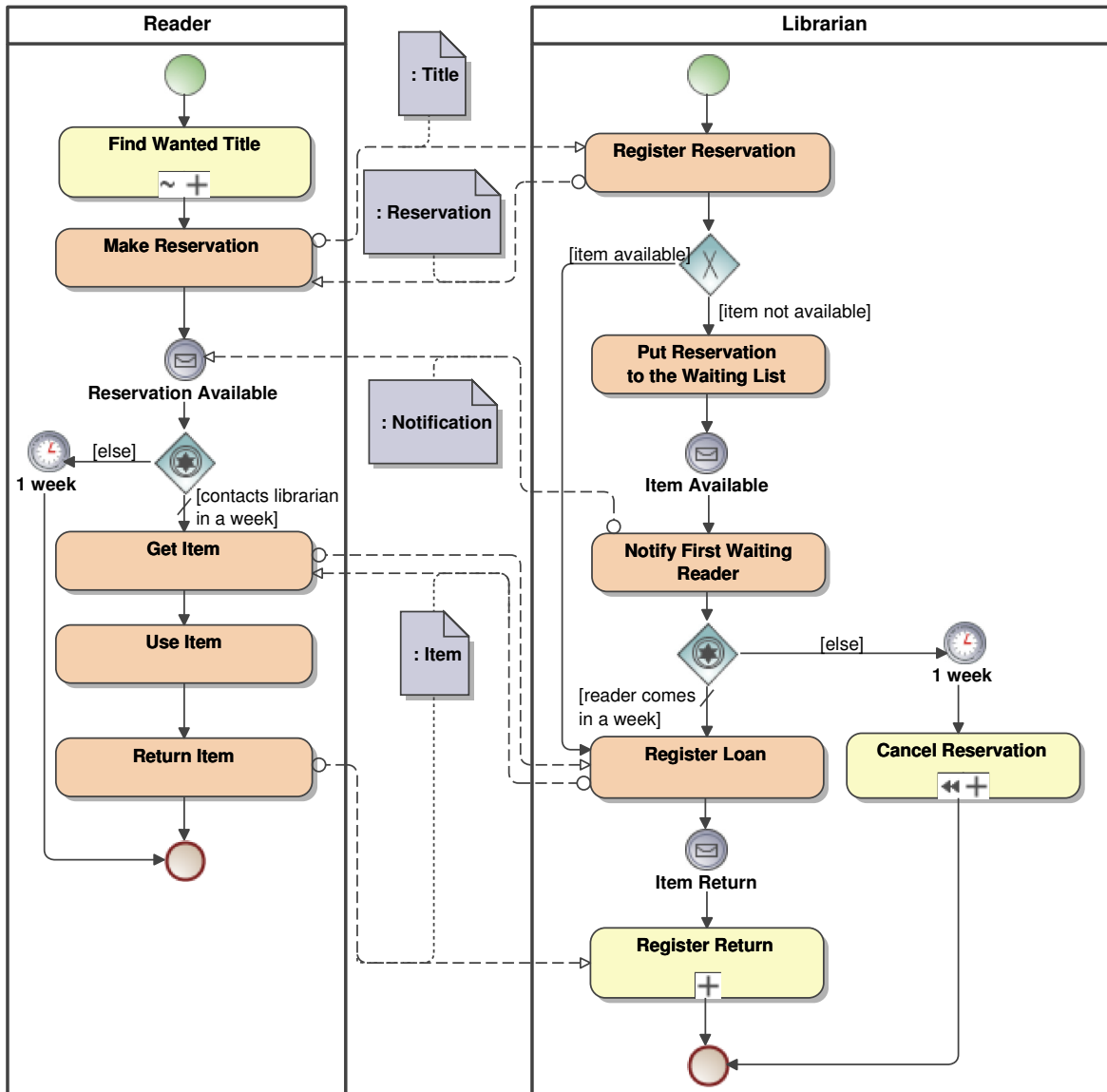


Figure 4. BPMN diagram visualizing business process of *Loaning item*.

The business processes are usually modeled in two forms: “*as is*”, representing current situation, and “*to be*”, representing target situation that should be reached after automation or refactoring. For software developers it is important to know which parts in target business process(-es) the software system should implement or support. The first step in moving from domain analysis to requirements definition is use case analysis. We propose to do use case analysis in the following steps:

1. Identify actors and group them into primary (main users), secondary (administration, maintenance, support), external systems, and pseudo (e.g. time).
2. Define main system use cases in a sketch use case diagram.
3. Group the created use cases into packages according to their coherence.
4. Generate actor – use case association matrix, giving compact overview of the whole model.

5. Prepare use case package details diagram, showing package use cases, their associations with actors and relationships between use cases including uses cases from different packages.
6. Prepare activity diagrams visualizing scenarios of complex use cases. In model, the activities should be nested within appropriate use cases and assigned as their behaviors.
7. Describe use cases according to pre-defined templates, e.g. *Rational Unified Process* use case document.

	Librarian	Reader	Time	Title Info System	User
[-] User Management	1				
[-] Requests	2	1			
[-] System Access					2
[-] Communications		1	3		
[-] Inventory Management	8				
[-] Loaning	3	2		1	1
Find Title					
Make Title Reservation					
Penalize Reader					
Register Item Loan					
Register Items Return					
Review Reader Profile					

Figure 5. Actor-Use Case matrix, automatically generated by MagicDraw UML from model content.

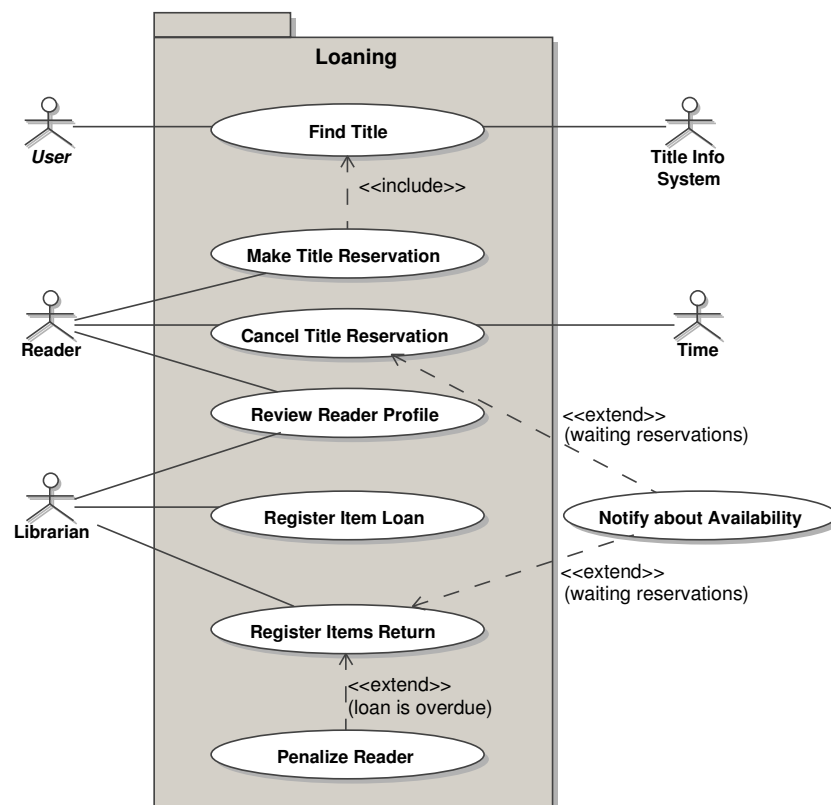


Figure 6. UML use case diagram for *Loaning* package.

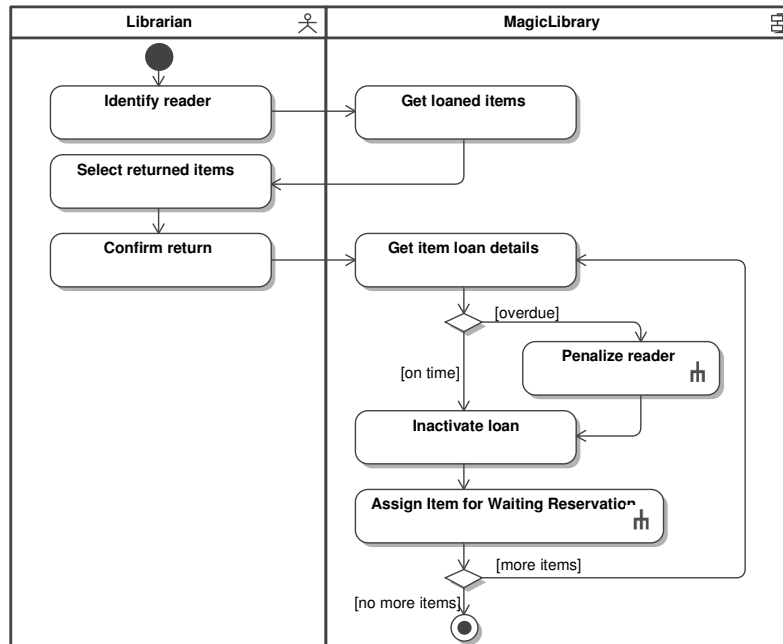


Figure 7. UML activity diagram representing behavior scenarios for use case *Register Items Return*.

Use case diagram captures only functionality that the end-user needs from the system. The other requirements such as non-functional requirements or detailed functional requirements are not captured in standard UML diagrams. The simplest way is to describe those in simple textual format and include references to use cases, their scenarios, etc. Another approach is to create specific UML extension for requirements modelling, i.e. introduce stereotypes for each important requirement type with tags for that requirement specific information and define types of links for tracing requirements, e.g. derive, satisfy, support.

Another aspect on which system analysts work in some projects is definition of data structure. It can be done using conventional UML class diagrams. If necessary, UML object diagrams can also be used for defining samples for explanation or testing of data structure defined in UML class diagrams. Since the focus here is on data structure, class operations compartments can be hidden in the diagram. Comparing to conceptual analysis, more elements are used here: attributes and association end specifications, enumerations, and generalization. Although such model is considered to be part of design, in practice quite often it is created and maintained by system analysts.

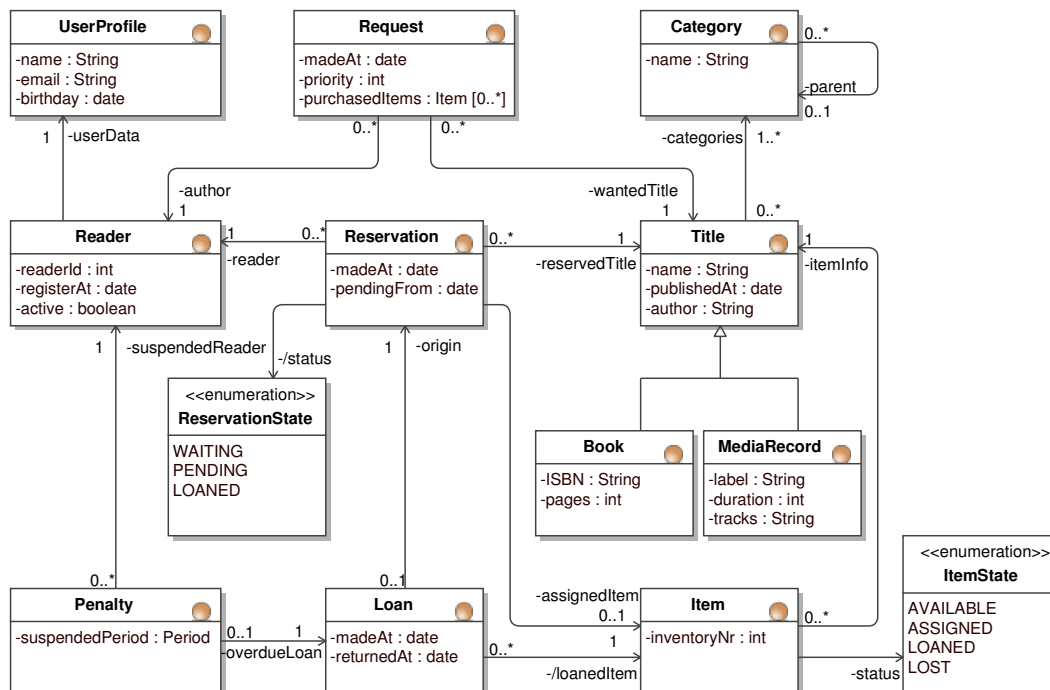


Figure 8. UML class diagram representing data structure.

For data-centric applications, it is very important to do data-flow diagrams showing information flows between different classifiers, e.g. system-context diagram indicates information flows from system to outside world entities, i.e. actors or external systems that need to be integrated.

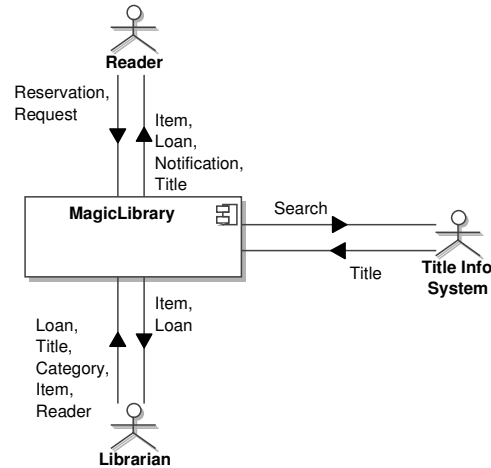


Figure 9. UML information flows diagram representing system context.

The last requirements modeling artifact for which system analyst might be responsible is user interface prototypes. The prototype itself can theoretically be mapped to *UML Composite Structure* diagram. However, when focusing on separate screen prototypes, people sometimes loose the big picture – what screens can be used by each actor and what are the possibilities to navigate from each screen to the other screens. For capturing this information, you can create GUI navigation map (should be done separately for each actor) using UML state diagram, where each state represent a screen, in which user is at the moment, and transition triggers represent GUI events, such as mouse double-click or clicking on some button. Again, this is considered a part of design, but in practice quite often falls on the shoulders of system analysts.

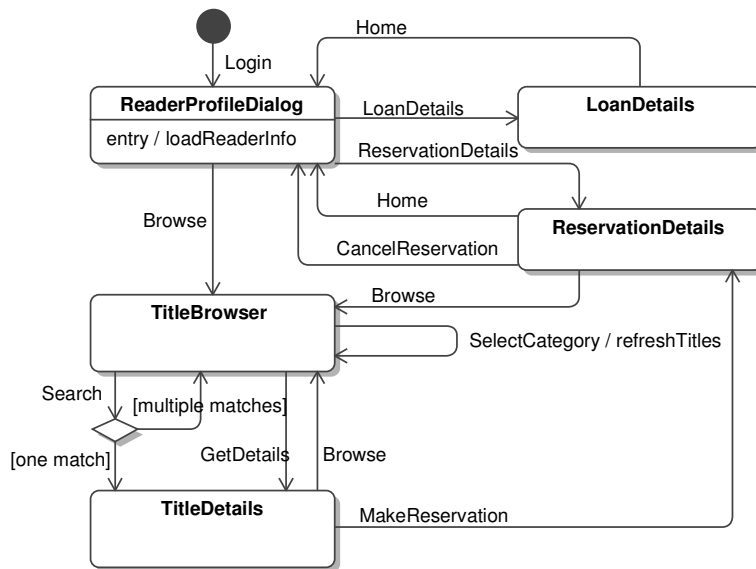


Figure 10. User interface navigation map for *Reader* actor.

Finally, we want to emphasize that the requirements analysis work should be iterative and incremental. Also, the ordering of modeling tasks might be different based on taken approach, or some steps might be omitted if not relevant to particular software development project.

4. Conclusions

We have presented a short review of modern requirement analysis issues emphasizing motivation for more consistent application of UML for requirements modeling. We have shown that the major requirements artifacts described in requirements engineering literature can easily be mapped to elements of UML. Finally, we described a conceptual framework for UML-based requirements modeling with illustrated modeling samples for library management system.

In the future, we plan to work on more detailed guidance for requirements modeling framework and develop a demo version of library management system for demonstrating the power of UML and model-based development approach.

References

- [1] **Rubinstein D.** Standish Group Report: There's Less Development Chaos Today. *SD Times*, March 1, 2007.
- [2] **Aranda J., Easterbrook S., Wilson G.** Requirements in the wild: How small companies do it. *15th IEEE International Requirements Engineering Conference (RE 2007)*, pp. 39-48
- [3] **Panis M. C., Pokrzywa B.** Deploying a System-wide Requirements Process within a Commercial Engineering Organization. *15th IEEE International Requirements Engineering Conference (RE 2007)*, pp. 295-300
- [4] **Object Management Group.** Unified Modeling Language: Superstructure. Formal Specification, version 2.1.2, 2007.
- [5] **Engels G., Heckel R., and Sauer S.** UML – A Universal Modeling Language? In *M. Nielsen, D. Simpson (Eds.): ICATPN2000, LNCS 1825*, pp. 24-38, 2000.
- [6] **Jacobson I.** Object-Oriented Software Engineering. Addison Wesley Professional, 1992.
- [7] **Wieggers K.** Software Requirements. 2nd edition, Microsoft Press, 2005.
- [8] **Object Management Group.** Business Process Modeling Notation Specification. Final Adopted Specification, version 1.0, 2006.
- [9] **Noran O.** UML vs. IDEF: An Ontology-oriented Comparative Study in View of Business Modelling. *Proceedings of International Conference on Enterprise Information Systems, ICEIS 2004*, Porto, 2004.
- [10] **Chen P. P.-S.** The entity-relationship model – toward a unified view of data. *ACM Transactions on Database Systems (TODS)*, vol. 1 (1), 1976.
- [11] **van Lamsweerde, A.** Goal-Oriented Requirements Engineering: A Guided Tour. *RE'01 – International Joint Conference on Requirements Engineering*, Toronto, 2001, pp.249-263.
- [12] **Penker M., Eriksson H.-E.** Business Modeling With UML: Business Patterns at Work. Wiley, 2000.
- [13] **Object Management Group.** Systems Modeling Language. Formal Specification, version 1.0, 2007.
- [14] **Gottesdiener E.** The Software Requirements Memory Jogger: A Pocket Guide to Help Software and Business Teams Develop and Manage Requirements. GOAL/QPC, 2005.
- [15] **Glinz M.** Problems and Deficiencies of UML as a Requirements Specification Language. *10th International Workshop on Software Specification and Design*, 2000, p.11 - 22
- [16] **Konrad S., Goldsby H., Lopez K., Cheng B.H.C.** Visualizing Requirements in UML Models. *International Workshop REV'06: Requirements Engineering Visualization*, 2006.
- [17] **Behrens H.** Requirements Analysis and Prototyping using Scenarios and Statecharts. *Proceedings of ICSE 2002 Workshop: Scenarios and State Machines: Models, Algorithms, and Tools*, 2002.
- [18] **Pinheiro da Silva, P.; Paton N. W.** UMLi: The Unified Modeling Language for Interactive Applications. Evans A.; Kent S.; Selic B. (Eds.): *UML 2000 – The Unified Modeling Language. Advancing the Standard*, pp. 117-132, Springer Verlag, 2000.
- [19] **Correa N., Giandini R.** A UML Extension to Specify Model Refinements. *XXXII Latin American Conference on Informatics (CLEI 2006)*
- [20] **Yu E. S. K.** Towards Modelling and Reasoning Support for Early-Phase Requirements Engineering. *Proceedings of the 3rd IEEE Int. Symp. on Requirements Engineering (RE'97)*, 1997.
- [21] **Butkienė R.** Method for Specification of Functional Requirements for Information System. Doctoral Dissertation, Kaunas University of Technology, 2002
- [22] **Castro J., Kolp M., and Mylopoulos J.** Towards Requirements-Driven Information Systems Engineering: The Tropos Project. *Information Systems*, Elsevier, 2002
- [23] **Šilingas D., Vitiutinas R.** Towards UML-Intensive Framework for Model-Driven Development. *Proceedings of CEE-SET 2007*