UML4PF – A Tool for Problem-Oriented Requirements Analysis

Isabelle Côté, Maritta Heisel, and Holger Schmidt
Software Engineering Group, University Duisburg-Essen
Duisburg, Germany
{*isabelle.cote,maritta.heisel,holger.schmidt*}*@uni-due.de*

Denis Hatebur
ITESYS Inst. f. tech. Sys. GmbH
Dortmund, Germany
d.hatebur@itesys.de

***Keywords*-Problem frames, UML profile, Tool support**

I. Introduction

We present a tool called *UML4PF*. This tool supports requirements analysis according to an enhanced version of Michael Jackson's problem frame approach [1]. Problem frames are patterns classifying software development problems. They pay special attention to the environment in which the software (called *machine*) will operate. That environment is represented by means of a *context diagram* that shows how the environment is structured in terms of *problem domains* and how the machine can interact with its environment. Interaction between domains is modeled by considering *shared phenomena*, which are controlled by only one domain and can be observed by other domains. Requirements are optative statements that *refer* to one or more problem domains and *constrain* at least one problem domain. Annotating (parts of) context diagrams with requirements yields *problem diagrams*. *Problem frames* are abstracted versions of problem diagrams. A simple subproblem of a more complex software development problem can be *fitted* to a problem frame by instantiating the frame diagram accordingly. Problem frames substantially support developers in analyzing problems to be solved. They show what domains have to be considered, and what knowledge must be described and reasoned about when analyzing a problem in depth.

To provide tool support for frame-based problem analysis, we first have carried over Jackson's original notation to UML by defining a corresponding profile, see [2]. We developed an Eclipse-plugin that allows software engineers to work with the defined profile. Furthermore, we developed a large number of *validation conditions* that make it possible to perform semantic checks on the developed diagrams. These validations conditions either refer to single diagrams (e.g., requirements are not allowed to constrain the machine), or they allow one to check the coherence between different diagrams (e.g., the messages of a sequence diagram must be phenomena of the corresponding problem diagram).

In this way, UML4PF supports software engineers in developing a coherent and complete set of requirements documents. Moreover, it supports the systematic development of an appropriate software architecture, see [3]. Next, we describe the technical realization of UML4PF. Then, we illustrate how to work with the tool, and finally, we discuss related work.

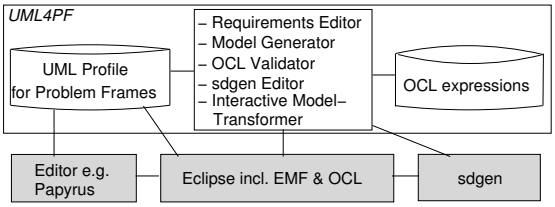

Figure 1: Tool Realization Overview

II. UML4PF Realization

UML4PF[1] consists of a UML[2] *profile* with formal validation conditions expressed in OCL[3] and an *Eclipse*[4] *plugin*. Figure 1 provides an overview of the context of our tool. Gray boxes denote re-used components, whereas white boxes describe those components that we created. The functionality of our tool comprises the following:

- The *UML Profile for Problem Frames* defines the relevant stereotypes for our approach, e.g., ≪`ProblemDiagram`≫ (see Fig. 3).
- The *Requirements Editor* allows one to add new requirements.
- The *Model Generator* automatically generates model elements, e.g., it generates observed and controlled interfaces from association names.
- The *OCL Validator* checks if the model is valid and consistent by evaluating our *OCL expressions*. It also returns the location of invalid parts of the model. All in all, we have defined about 50 OCL validation conditions for the analysis phase. The time needed for checking only depends on EMF[5] and is about 0.5 seconds per validation condition. The influence of the model size on the checking time is less than linear.
- The *sdgen Editor* is used to edit sequence diagrams.
- The *Interactive ModelTransformer* serves to create software architectures through interactive model transformations.

The tool UML4PF is still under development and evaluation.

[1] http://swe.uni-duisburg-essen.de/en/research/tool
[2] http://www.uml.org/
[3] http://www.omg.org/spec/OCL/2.0/
[4] http://www.eclipse.org/
[5] http://www.eclipse.org/emf/

III. Working With UML4PF

To illustrate how to work with the tool, we consider a simple software development subproblem that is part of the larger task of developing an online vacation rentals system. The vacation rentals system shall allow a potential guest to browse and book available holiday offers. Staff members are responsible for recording the incoming payments. Furthermore, they make new holiday offers available (req. R01).

We create a new project with a new model both named *VacationRentals* and apply our UML profile for Problem Frames to the model using a EMF-compatible UML editor such as Papyrus[6].

Problem elicitation and description: We describe the intended environment of our vacation rentals system by a context diagram. This is achieved by using the graphical elements provided by Papyrus' editor: first we place a package in a class diagram tab, which we name "VacationRentals_env". All other necessary domains and associations are then placed within this package. The resulting context diagram is depicted in Fig. 2. Domains are represented as classes, and interfaces between domains – containing sets of shared phenomena – are represented as associations. The package is annotated with the stereotype ≪ContextDiagram≫.

Next, we execute the *model generator*. This can be done by right-clicking on the uml-file and selecting the entry "Model Generator" in the context menu. After the generation, all interfaces corresponding to connections in the context diagram exist.

To check the consistency of the context diagram, we right-click on the uml-file, this time selecting the entry "Validate now". The conditions are grouped by the process step they can be applied to. It is possible to select, which step(s) shall be executed. The validation conditions are checked, and the results are displayed (see Fig. 5). Fulfilled validation conditions are displayed in green, violated ones in red. For the violated conditions, we provide further expressions that indicate which elements cause the condition to fail. These conditions are displayed in light-gray. All checks concerning the context diagram pass.

Problem decomposition and classification: We now decompose the overall problem into subproblems. For each subproblem, we create a package with the stereotype ≪ProblemDiagram≫. We re-use the classes of the context diagram where applicable. Furthermore, we add classes with the stereotype ≪Requirement≫ and assign the appropriate stereotype to the dependencies originating from this class. One of the resulting subproblems is given in Fig. 3.

After completing the decomposition, we again execute the model generator and the validator. One condition has been violated (see bottom of Fig. 5). The error diagram causing this violation is the dependency between the requirement and the domain guest has the stereotype ≪constrains≫. This is not allowed, as we cannot constrain actions of users. Therefore, we must replace the ≪constrains≫-dependency by a ≪refersTo≫-dependency. Additionally, UML4PF allows us to check if a described problem diagram is an instance of a pattern (see [2]).

Derive Software Specification: In this step, we draw sequence diagrams. The *sdgen* editor can be accessed by right-clicking on the uml-file containing the EMF-model and selecting the menu entry "open with" followed by "sdgen Editor". For each problem diagram we create sequence diagrams capturing the normal as well as exceptional behavior. The domains in the problem diagram are directly linked to the machine become lifelines in the respective sequence diagram. The operations in the interfaces are transferred to messages between the corresponding lifelines. Fig. 4 shows the sequence diagram for the subproblem Make.

In the subsequent steps we

- *derive the technical context diagram*. This is a special context diagram showing the technical details of the machine environment, e.g., the used web servers. We can check the consistency of context diagram and technical context diagram.
- *Specify operations and data structures*. The operations are specified using OCL-expressions. The syntax of these expressions is checked via the built-in OCL parser. Furthermore, UML4PF can check for completeness of the operations against the sequence diagrams.

IV. Related Work

Charfi et al. [4] use a modeling framework called *Gaspard2* to design high-performance embedded systems-on-chip. We have been inspired by this approach. However, their approach does not support problem frames. Other important UML profiles are *SysML*[7] for system engineering and *MARTE*[8] for model-driven development of real time and embedded systems. The UML profile for MARTE supports specification, design, and verification/validation stages.

We are not aware of other tools supporting the work with problem frames on the semantic level, as does UML4PF.

References

[1] M. Jackson, *Problem Frames. Analyzing and structuring software development problems*. Addison-Wesley, 2001.

[2] D. Hatebur and M. Heisel, "Making Pattern- and Model-Based Software Development More Rigorous," in *Proc. of 12th Int. Conf. on Formal Engineering Methods (ICFEM)*, J. S. Dong and H. Zhu, Eds. Springer, 2010.

[3] C. Choppy, D. Hatebur, and M. Heisel, "Systematic architectural design based on problem patterns," in *Relating Software Requirements and Architectures*. Springer-Verlag, 2011.

[4] A. Charfi, A. Gamatié, A. Honoré, J.-L. Dekeyser, and M. Abid, "Validation de modèles dans un cadre d'IDM dédié à la conception de systèmes sur puce," in *4èmes Jounées sur l'Ingénierie Dirigée par les Modèles (IDM 08)*, 2008.

[6] http://www.papyrusuml.org

[7] http://www.omgsysml.org/

[8] http://www.omgmarte.org/

APPENDIX

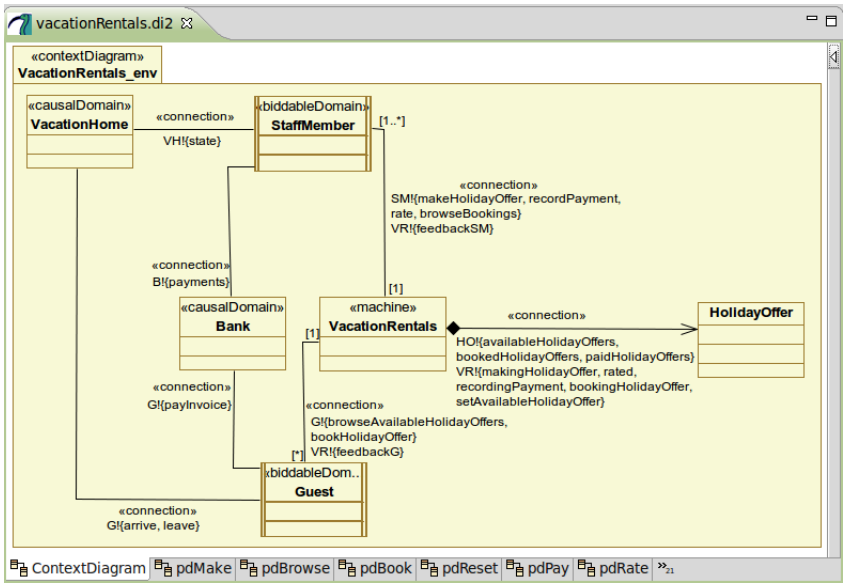

Figure 2: Screenshot Vacation Rentals: Context Diagram

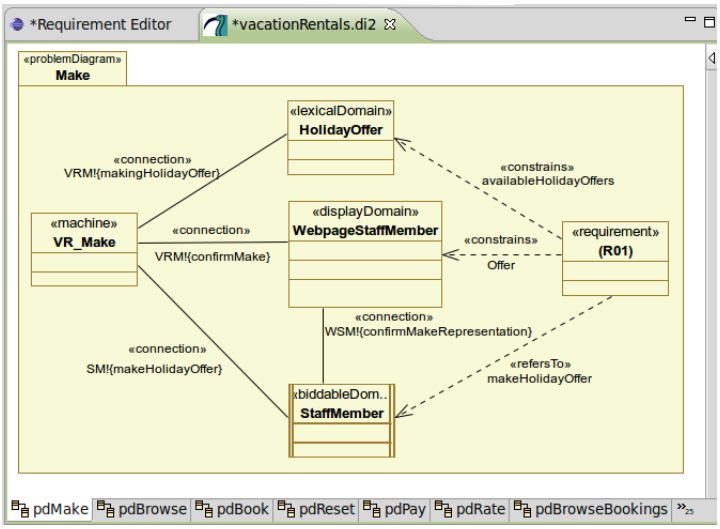

Figure 3: Screenshot Vacation Rentals: Problem Diagram for R01

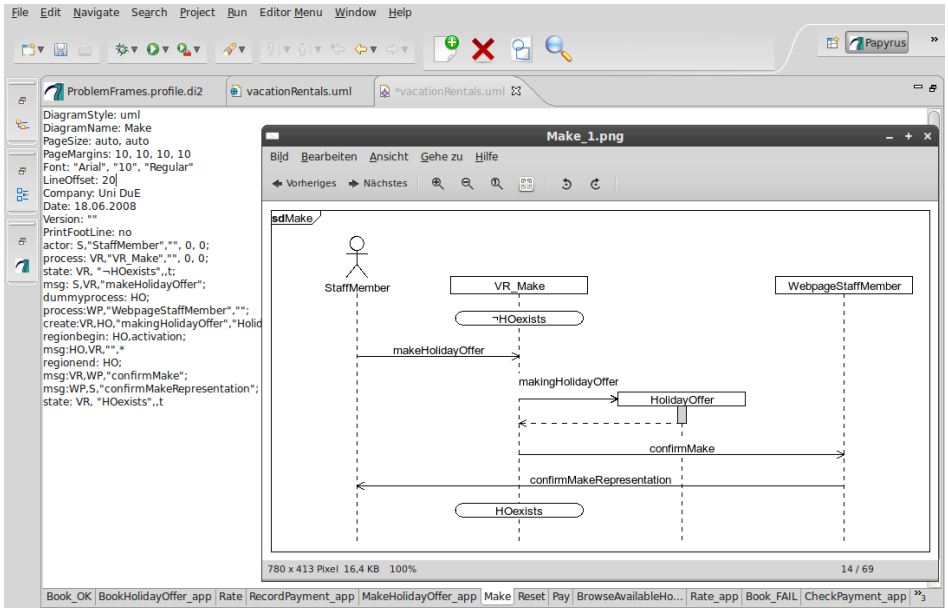

Figure 4: Screenshot Vacation Rentals Sequence Diagram for Subproblem Make (R01)

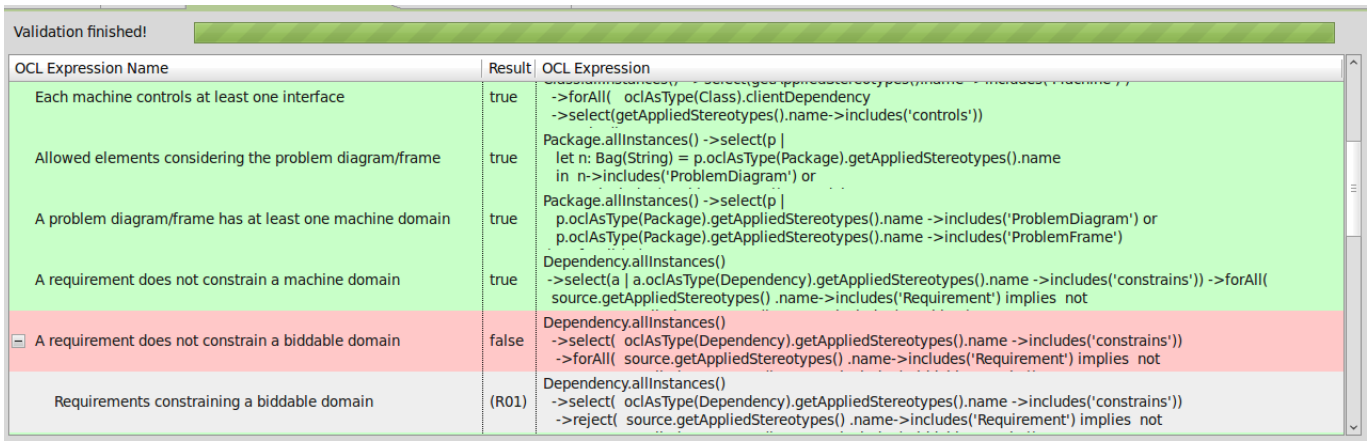

Figure 5: Screenshot of OCLValidator