Embedded Software Modeling using UML2: A Case Study

Eliane Siegert, Milena Marques, Lisane Brisolar
{esiegert, mrsmarques, lisane}@inf.ufpel.edu.br

Centro de Desenvolvimento Tecnológico
Universidade Federal de Pelotas

Abstract

UML is a modeling language usually adopted in the software design and which can be used to accelerate the software development through transformations of models. The goal is to model whole system providing abstraction for designers. With UML2 resources is possible to represent different views of the system. This paper presents a case study, in which an embedded system is represented by UML2 models. Class and sequence diagrams are used to give structural and behavioral views of the system, thus helping developers to quickly build a complete view of the system. Moreover, from these models, code can be automatically generated, including structural and behavioral parts.

1. Introduction

Embedded software designers should deal with applications increasingly complex and work to produce high quality products, with low cost and short delivery time. So, there is a strong motivation to adopt new approaches and methodologies able to provide high abstraction and at same time to speed up the embedded software development process.

Model-driven engineering approaches (MDE) [1] provide automation and abstraction in system design. This paradigm considers models like central artifacts of software design being used for all engineering phases, not only in the documentation. Models are abstractions that facilitate the understanding of the system and help its building.

The UML [2] is the standard language for the software modeling, providing high level of abstraction and a variety of diagrams that are used to represent different views of a system. The last version of the language, UML 2, offers many resources for modeling of system behavior using sequence diagrams [3][4], enabling the representation of a more completed behavioral view. This paper presents a case study of the embedded software modeling using UML2, which allows evaluate the use of its notations in this specific domain. From this model, object-oriented code can be generated using appropriate tools, according the principles of model-driven engineering [5].

The paper is organized as following. Section 2 presents the methodology adopted for the model construction. Section 3 presents the case study, discussing the main diagrams used to model the system. Section 4 discusses the evidences observed in the case study and Section 5 presents the main conclusions and future works.

2. Methodology

First, a use case diagram was defined, and from this functional view, the structural and behavioral views were constructed. The class diagram specifies the main static elements of the system, such as classes and interfaces as well their relationships. A class diagram was used to represent the structural view of the system, while several sequence diagrams are used to give the behavioral view of the system. Foremost, a main sequence diagram was defined, which represents an overview of the whole system behavior. To detail this view, other sequence diagrams were used which represent the behavior enclosed by a method call or for a model fragment (indicated by Ref, a UML2 resource). In these diagrams, the message exchange between methods, loops, conditionals or references for the others sequence diagrams are used to represent the expected behavior. A tool called Papyrus [6] was used for the construction of the UML2 diagrams.

3. Case Study

The case study consists in the modeling of an embedded software for a washing machine system. The application was chosen mainly because it is a simple example of embedded software and therefore easy to understand. This section presents and discusses the main diagrams used for the system modeling. For space limitations, some diagrams utilized have been omitted.

The structural view of the system was modeled using the class diagram illustrated in fig. 1. Basically, this model is composed of seven concrete classes (WashingMachine, Timer, WashOption, Engine, DoorSensor, WaterSensor, and TempSensor), an abstract class named Sensor and an interface called Machine. The classes
Engine and WashingMachine implement the interface Machine. The abstract class Sensor has the concrete classes DoorSensor, TempSensor, and WaterSensor as subclasses. In this model, it is possible to observe also association between WashingMachine and the classes Engine, WaterSensor, WashOption, and Timer. The WashingMachine is the main class of the system, which has the method main, representing the starting point of the system, in addition to other methods (wash, rinse, spin, fill, and drain) that represent the features of the system.

From the class diagram, sequence diagrams were built to represent the system behavior, defined by a set of interactions among objects. Fig. 2 illustrates a sequence diagram (sd) representing the method main of the class WashingMachine, and named sd Main to indicate that this diagram corresponds to this method. In this diagram, the interaction between objects of the class WashingMachine and WashOption are represented.

Firstly, the washing machine must identify the selected operation. For this, the object WashingMachine invokes the method getWashSelection from object washOption. This method returns an integer value used to represent the selected option. Therefore, the return value is checked for the Alt fragment (UML2 resource), which represents a selection of two or more possible paths, and determines what operation must be performed by the machine. In this case, if option “1” is selected, then the operation StandardWash is performed, if “2” the operation TwiceRinse and, if “3” the operation Spin. In this diagram, to reduce complexity, each operation mode was represented by a method invocation. However, each method can enclose others method invocations or other significant behavioral fragments, and in this case, another sequence diagram can be used to detail the method.
From the main sequence diagram (sd Main), other diagrams are used to detail the behavior, one for each invoked method. The diagram shown in Fig. 3 represents the behavior enclosed by the method spin (from Fig. 2), which corresponds to the detailed behavior of the operation Spin. This operation starts when the object WashingMachine invokes the method turnOn() of the object engine, followed by the invocation of the method setDuration of the object timer, passing the period of time that the machine must be turned on. The execution time of the washing machine is detailed in another sequence diagram, here abstracted by the Ref Period.

![Sequence diagram: Behavioral view of the spin interaction.](image)

Figure 4 illustrates the sd Period, detailing the interaction Period referenced in the sd Spin (Fig. 3) and by others sequence diagrams which are not shown in this paper. The control of the duration of each operation is represented by this sequence diagram. To initialize the count by the timer, the object washMachine invokes the method start. The method count should be invoked several times in order to increment the current value of the object timer. This behavior is modeled as a loop where this invocation is repeated while current time is different from the established duration of the task, as indicated by the logical condition “[time!=duration]”.

![Sequence diagram: interaction Period](image)

Through the references used to connect the diagrams, it is possible to represent whole system behavior, in which each sequence diagram shows a part of the system actions. In addition, the set of these diagrams composes the behavioral view of the system, which can be detailed until the method call level.

4. **Analyzing the evidences**

The case study demonstrates that UML provides a set of modeling notations that allow represent different views of a system, thus facilitating the representation of several aspects of the design, such as functional, structural, and behavioral views using use case, class, and sequence diagrams, respectively. The use of UML2 resources (as loop, alt, opt, and ref) enables the construction of a complete behavioral view and enables a more
complete automatic code generation. In conjunction with a model-driven engineering approach, the use of UML2 models could speed up the system development, since model transformations can be used to obtain the finished product more quickly.

However, UML2 continues presenting limitations for the modeling of embedded systems [7]. This language does not allow specifying non-functional requirements (as timing and energy requirements) commonly important in an embedded system design. For that reason, should be used stereotypes defined by UML extensions, such as MARTE [8] and SysML [9] specially defined by the embedded community.

5. Conclusions and future work

This paper presents a case study of modeling software for embedded systems using features supported by the UML2. The software used in the case study was the control of a washing machine using basically class and sequence diagrams. This case study also was conspicuous to build a model from where most part of the code could be automatically generated. For that reason, several sequence diagrams were elaborated for representing whole system behavior using resources from UML2. Although details are inserted into the model, it continues to be easier to be built than code, since it does not include so many details as a program. Thus, we conclude that the use of UML2 models can facilitate and speed up the embedded software design process. As future work, we plan to study the UML extensions proposed for the embedded domain and conduct a new case study to evaluate the use of stereotypes and new diagrams provided by these extensions.

6. References


