On the Modular Specification of Non-Functional Properties in DSVLs

Javier Troya, Antonio Vallecillo, and Francisco Durán

GISUM/Atenea Research Group. Universidad de Málaga (Spain)
{javiertc,av,duran}@lcc.uma.es

Abstract. In previous work we have presented an approach to monitor non-functional properties of systems modeled in terms of domain specific visual languages using *observers*. In this work we present an approach to decouple the definition of observers behavior and systems behavior. Having a library with different kinds of observers behavior, and having the behavioral definition of the system, weaving links can be established among them in order to include observers in the system behavioral specification.

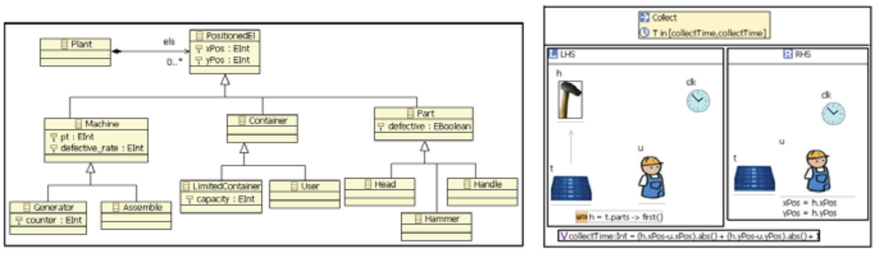
Key words: DSVLs, weaving mechanisms, observers

1 Introduction

Domain specific visual languages (DSVLs) play a cornerstone role in Model-Driven Engineering (MDE) for representing models and metamodels. The benefits of using DSVLs is that they provide an intuitive notation, closer to the language of the domain expert, and at the right level of abstraction. The Software Engineering community's efforts have been progressively evolving from the specification of the structural aspects of a system to modeling its behavioral dynamics. Thus, several proposals already exist for modeling the structure and behavior of a system. Some of these proposals also come with supporting environments for animating or executing the specifications, based on the transformations of the models into other models that can be executed [1, 2].

In previous work [3] we proposed our own approach to monitor non-functional properties of DSVLs. The idea is to integrate new objects, named *observers*, in the system specifications that capture such properties. Our proposal is based on the observation of the execution of the system actions and of the state of its constituent objects in the case of DSVLs that specify behavior in terms of rules. The use of observer objects enables the analysis of some of the properties usually pursued by simulation, like cycle-times, busy and idle cycles, mean-time between failures, throughput, delay, etc. The OMG, in turn, defines different kinds of observers in the MARTE specification [4], which are similar to ours. However, they cannot be used to describe requirements and constraints on models, as we do. Furthermore, we can use our observers to dynamically change the system behavior, in contrast with the more "static" nature of MARTE observers.

In our approach, users define their own observers, according to the non-functional properties they want to monitor. Then, these observers are to be included in the behavioral rules of systems. The resulting rules are difficult to maintain, since the observers



(a) Metamodel. (b) Behavioral Rule.

Fig. 1. Production Line System

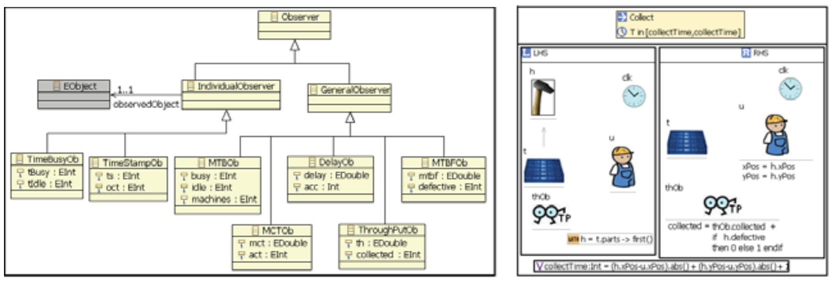
and systems specifications are mixed. In this paper we propose an approach to decouple the process of integrating observers in the systems specification. Having a library where different kinds of observers generic behaviors are available, the idea is to weave these behaviors with the systems behavioral rules. In this sense, our approach may seem similar to the Software Metrics Meta-model (SMM) [5] defined by the OMG, a metamodel for representing measurement information related to software, its operation and design. However, our approach is more flexible since new metamodels and metrics to monitor non-functional properties can be defined by the modeler. Furthermore, we give semantics for defining the dynamic behavior of systems and their non-functional properties.

After this introduction, Section 2 briefly describes our current approach for monitoring non-functional properties of systems. Then, in Section 3 we present our ideas for a modular specification of observers. Finally, Section 4 concludes the paper and outlines how we would like to continue this work.

2 Monitoring Non-Functional Properties of Systems using Observers

In this section we briefly describe our current approach for the specification and monitoring of non-functional properties of systems in *e-Motions* [6].

The first step is to define our DSVL. DSVLs are defined in terms of three main elements: abstract syntax, concrete syntax and semantics. The abstract syntax defines the domain concepts that the language is able to represent, and is defined by a metamodel. The concrete syntax defines the notation of the language, and in *e-Motions* it is defined by assigning an icon to each concept in the metamodel. The semantics describe the meaning of the models represented in the language, and in case of models of dynamic systems (such as ours) the semantics of a model describe the effects of executing that model. In our case, semantics are specified by a set of behavioral rules. Snapshots of a part of a metamodel and a behavioral rule for a particular example of a production line system are shown in Figure 1. Regarding the metamodel, Machines generate and assemble Parts and Containers either transport or keep these Parts. The rule models how



(a) Observers Metamodel. (b) Behavioral Rule with Observers.

Fig. 2. Observers for the Production Line System

a Hammer that is placed in a LimitedContainer is collected by a User. The complete description of this case study is presented in [7].

Once the system behavior has been specified, it is time to add observers to the rules. The first step is to identify which non-functional properties the user wants to monitor: throughput, mean-time between failures, delays, response times, etc. Observers are specified by means of a metamodel (an observers metamodel for the production line system is shown in Figure 2(a)), which is then combined with the system metamodel to be able to use the observers in the DSVL. In the observers metamodel it can be seen that we define observers of two types: *Individual* and *General*. The former are those which are attached to individual objects to monitor their state and/or behavior, while the latter monitor individual observers, as well as the remaining objects in the system, to build derived measures for the non-functional properties we want to monitor. The next step is to include observers in the behavioral rules in order to monitor non-functional properties of our DSVL. The observer added in Figure 2(b) is meant to monitor the throughput of the production line. Concretely, in this rule the observer updates its collected attribute everytime a new Hammer is collected by the User. The specifications of systems with observers are then translated into the corresponding formal specifications in Maude [8]. In fact, since the Maude rewriting logic specifications are executable, they can be used as a prototype of the system, which allows us to simulate and analyze it. After the simulation, observers contain information about how the system behaved in terms of its non-functional properties.

In our current approach, the addition of observers may require to change the existing behavioral models to a large extent, rendering in some cases a fairly complex, difficult to understand and potentially hard to maintain system models. This is because the addition of observers in the behavioral rules depends directly on the type of system we are dealing with. In this way, different kinds of observers must be defined for each system. Next section presents an approach to overcome these limitations.

3 A Modular Approach for the Specification of Observers

In this section we propose the use of aspect-oriented techniques, whereby a modular specification of the behavior of observers is provided, and then woven with the system behavioral rules. Although this approach limits the flexibility required for observers in some situations, it can be used for the majority of properties. It also provides a modular approach to the specification and monitoring of individual properties, for which observers can be independently defined and reused across system specifications. Thus, the idea is to define a library with different observers behavior, which can then be reused by concrete systems to incorporate observers within their behavior specifications. This way, observers metamodel and rules are defined only once and used with different kinds of systems.

Our approach is based on standard software measurement approaches, which define *base* and *derived* measures [9]. The former allow measuring individual object attributes, while the latter build on the values of base measures to define aggregated metrics. Similarly, we have our *individual* and *general* observers. Although of different nature, both kinds of observers can be specified using a similar approach. Their behavior, once specified, can be woven to the functional behavior of a system to produce the complete system specifications. Thus, the behavior of observers follows a regular pattern, that corresponds to their life-cycle: creation, monitoring and termination. Rules are defined for each of these phases. Due to the space limitation, in this paper we describe the generic behavior for general observers. The generic definition of the behavior of all the observers shown in Figure 2(a) can be found at [10].

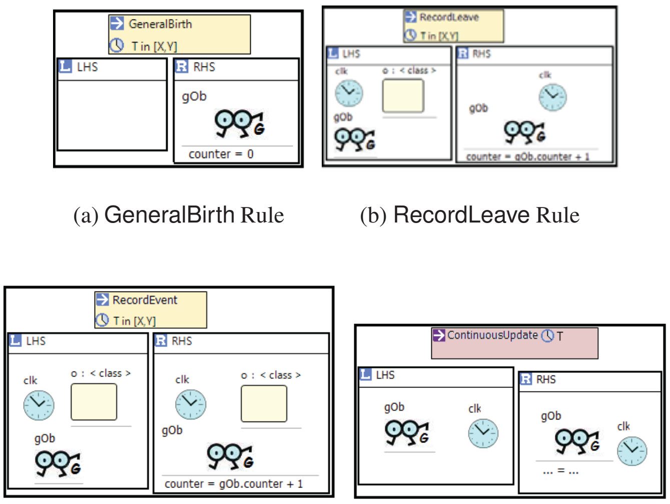
3.1 Generic Behavior of General Observers

The behavior of general observers is normally determined by four of rules. We show them here in general, and then these rules are specialized when defining the behavior of a concrete kind of observer (the behavior of all the observers of the production line system example can be found in [10]). The first rule (GeneralBirth, shown in Figure 3(a)) specifies the creation of a general observer. Since they are created at start time, this rule is normally woven to the initial rule of the system. Two rules specify how global observers update their state variables, depending on whether they do it when an object disappears from the system, or when the object participates in a rule. Thus, generic rule RecordLeave (Fig. 3(b)) shows how a counter attribute is updated when an object leaves the system. Similarly, rule RecordEvent (Fig. 3(c)) models the behavior of a global observer that records that an object has participated in a rule. Some attributes of some general observers need to be updated as time moves forward. Such a behavior is modeled with ongoing rules, whose generic form is shown in Fig. 3(d).

3.2 Weaving the Rules

Once we have the rules that specify the observers behavior independently from any concrete system specification, we need to weave them with the system rules. For that we use a weaving model that uses the AtlanMod Model Weaver (AWM[1]) to define the

[1] http://wiki.eclipse.org/AMW



(a) GeneralBirth Rule (b) RecordLeave Rule

(c) RecordEvent Rule (d) ContinuousUpdate Rule

Fig. 3. Generic rules for general observers.

correspondences between the *generic* objects in the observers rules and the concrete objects that appear in the system behavioral rules. In this case, the object in the generic rule and the one in the concrete rule that match will be woven, and the remaining elements in the generic rule will be copied to the concrete one. It is also possible to define a correspondence between one observer rule and one system rule. In this case, all the elements from the observer rule will be copied to the system rule. When defining the weaving links between the observers and the system rules it is possible to add expressions that overwrite how the values of the attributes are calculated in the observer rule. In this way we allow some kind of rule *parametrization*. Finally, it is possible (and very common) to establish correspondences between several observers rules and one system rule, when we want to add more than one kind of observer to a rule. Only one final rule is produced with the results of all weaves. To illustrate this approach, let us apply it to the production line system example. Starting from the behavioral rules without observers and assuming that we have defined the rules for the observers, in Figure 4 we present the matching for inserting the general ThroughPutOb observer in the Collect behavioral rule from Figure 1(b). The weaving link is defined between the Hammer object in the Collect rule of the system and the generic object in the RecordLeaveTP rule. The effect of this binding is to include the ThroughPutOb observer in both the LHS and RHS of the Collect rule, creating the rule shown in Fig. 2(b). Note that the expression used to calculate the value of the collected attribute in the RHS of the observer rule has been overwritten by a new expression (this is indicated in the figure inside the box between the two woven rules).

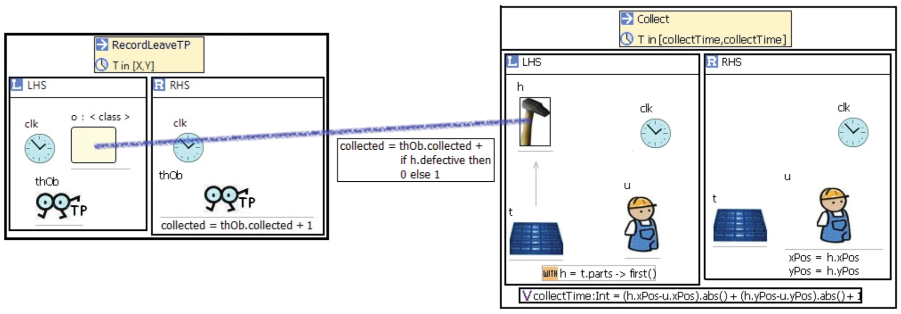

RecordLeaveTP
T in [X,Y]
LHS
clk
o : < class >
thOb
TP
RHS
clk
thOb
TP
collected = thOb.collected + 1
collected = thOb.collected + if h.defective then 0 else 1
Collect
T in [collectTime,collectTime]
LHS
h
clk
t
u
WITH h = t.parts -> first()
RHS
clk
u
t
xPos = h.xPos
yPos = h.yPos
V collectTime:Int = (h.xPos-u.xPos).abs() + (h.yPos-u.yPos).abs() + 1

Fig. 4. Weaving the RecordLeaveTP and Collect rules.

4 Conclusions and Future Work

We have described our current approach to specify and monitor non-functional properties of DSVLs. We have also presented how we want to extend that approach in order to include observers in the systems behavioral rules in a decoupled and modular way. The idea is to have a library with the behavior of different kinds of observers, and then apply weaves between these rules and the systems rules. Our plan for future work is to continue the study of this approach in order to completely implement it.

References

1. Efroni, S., Harel, D., Cohen, I.R.: Reactive animation: Realistic modeling of complex dynamic systems. Computer **38**(1) (2005) 38–47
2. Ermel, C., Ehrig, H.: Behavior-preserving simulation-to-animation model and rule transformations. ENTCS **213**(1) (2008) 55–74
3. Troya, J., Rivera, J.E., Vallecillo, A.: Simulating domain specific visual models by observation. In: Proc. of the Symposium on Theory of Modeling and Simulation (DEVS'10), Orlando, FL (US) (2010)
4. OMG: A UML Profile for MARTE: Modeling and Analyzing Real-Time and Embedded Systems. OMG. (2008)
5. OMG: Software Metrics Meta-Model (SMM). OMG. (2009)
6. Rivera, J.E., Durán, F., Vallecillo, A.: A graphical approach for modeling time-dependent behavior of DSLs. In: Proc. of VL/HCC'09, Corvallis, Oregon (US) (2009)
7. Atenea: Non-functional monitoring in the PLS case study (2011) `http://atenea.lcc.uma.es/index.php/Main_Page/Resources/E-motions/PLSObExample`.
8. Clavel, M., Durán, F., Eker, S., Lincoln, P., Martí-Oliet, N., Meseguer, J., Talcott, C.: All About Maude – A High-Performance Logical Framework. Number 4350. Springer, Heidelberg, Germany (2007)
9. García, F., Bertoa, M.F., Calero, C., Vallecillo, A., Ruíz, F., Piattini, M., Genero, M.: Towards a consistent terminology for software measurement. Information and Software Technology **48**(8) (2006) 631–644
10. Atenea: Modular Approach in the PLS case study (2012) `http://atenea.lcc.uma.es/index.php/Main_Page/Resources/E-motions/PLSObModExample`.

Secure Business Process model specification through a UML 2.0 Activity Diagram profile

Alfonso Rodríguez[1], Eduardo Fernández-Medina[2], Juan Trujillo[3], and Mario Piattini[2]

[1] Computer Science and Information Technology Department,
University of Bio-Bio, Chillán, Chile
alfonso@ubiobio.cl
[2] GSyA Research Group, Information Systems and Technologies Department,
University of Castilla-La Mancha, Ciudad Real, Spain
{Eduardo.FdezMedina, Mario.Piattini}@uclm.es
[3] LUCENTIA Research Group, Department of Software and Computing Systems,
University of Alicante, Alicante, Spain
jtrujillo@ dlsi.ua.es

1 Summary

Business processes are currently important resources both for enterprises' performance and to enable them to maintain their competitiveness. In recent years, the languages used for business process representation have been improved and new notations have appeared. In spite of the fact that the importance of business process security is accepted, the business analyst perspective in relation to security has so far scarcely been dealt with. Moreover, security requirements cannot be represented in modern business process modeling notations.

In this paper, we have presented an extension of the UML 2.0 Activity Diagram (UML 2.0-AD) which allows us to specify security requirements in the business process domain. This specification corresponds with a Computation Independent Model (CIM) within the MDA (Model Driven Architect) approach. We have based our proposal on UML 2.0-AD for three main reasons: (i) UML 2.0 description significantly improves the business process representation through Activity Diagrams, (ii) UML can easily be extended, thus allowing it to be tailored to a specific domain and (iii) UML modeling is dominant in the software industry and this eases the transformation of business process models into models which are closer to implementation.

In our proposal, called BPSec (Business Process Security), we use the approach driven by models, MDA, because it establishes that a business process corresponds to a Computation Independent Model, while UML artifacts such as analysis classes and use cases correspond to platform independent models. Thus, a model transformation from a Secure Business Process (SBP) model to analysis classes and use cases is the transformation from CIM to Platform Independent Model (PIM), according to the MDA paradigm.

In Figure 1, we show all the details of our proposal. We have colored the following elements in dark grey: (i) BPSec; the UML 2.0-AD extension presented in details in this work, (ii) M-BPSec, a method that we have designed for the ordered and