A Tool for Verified Design using Alloy for Specification and CrocoPat for Verification

Alain Wegmann, Lam-Son Lê, Lotfi Hussami
Ecole Polytechnique Fédérale de Lausanne (EPFL)
School of Computer and Communication Sciences
Lausanne, Switzerland
{Alain.Wegmann, LamSon.Le}@epfl.ch

Dirk Beyer
Simon Fraser University
School of Computing Science
Surrey, B.C., Canada

ABSTRACT

The context of our work is a project that focuses on methods and tools for modeling enterprise architectures. An enterprise architecture model represents the structure of an enterprise across multiple levels, from the markets in which it operates down to the implementation of the technical systems that support its operation. These models are based on an ontology that defines the model elements and their relations. In this paper, we describe an efficient method to fully automatically verify the design that our modeling tool manages. We specify the ontology in Alloy, and use the efficient interpreter for relational programs CrocoPat to verify that the design fulfills all constraints specified in the ontology. Technically, we transform all constraints from Alloy into a relational program in CrocoPat's programming language. Then, we execute the relational program and feed it with a relational representation of the design as input, in order to check that the design element instances fulfill all constraints of the Alloy representation of the ontology. We also present the current limitations of our approach and how –by overcoming these limitations– we can develop an Alloy-based parameterized modeling tool.

Categories and Subject Descriptors: I.6.4 [Simulation and Modeling]: Model Validation and Analysis; D.2.4 [Software Engineering]: Software/Program Verification – V*alidation*; J.6 [Computer Application]: Computer-Aided Engineering – *Computer-aided design (CAD)*;
General Terms: Design, Languages, Verification
Keywords: Design Validation, Alloy, CrocoPat, Enterprise Architecture Modeling, Design Constraint Checking

1. INTRODUCTION

IT and business alignment is one of the top-ranked issues for Chief Information Officers (CIO) [1]. Enterprise architecture (EA) addresses this alignment issue. EA deals with the specification and design of systems that span from business (e.g., market, companies, departments, employees) down to technology (e.g., technical systems, applications, software components). In our previous work, we developed the EA modeling method *SEAM* (*Systemic Enterprise Architecture Methodology*) [2] and the corresponding tool implementation *SeamCAD* [3]. When using SEAM, the EA team develops an enterprise model that represents the markets and value networks in which a company is active, the company, the employees of the company, the technical systems identified in the company as well as their components. Each of these entities is considered as a *system*, in the general sense, and these systems are considered in different organizational levels. The enterprise model is graphically represented with a notation similar to UML 1.5.

The EA model (also called design) is based on an ontology (also called meta-model) that makes systemic concepts (such as contexts or boundaries) explicit and that represents, in a systematic manner, the systems identified at the different levels. This ontology is based on the foundation modeling concepts as defined in Part 2 of the ISO/ITU *Reference Model of Open Distributed Processing (RM-ODP)* [4]. We have formalized the RM-ODP Part 2 in Alloy[1] [5], and then refined the Alloy model to be able to represent hierarchical systems [6, 7]. The SEAM tool implements this refined version of the ontology.

In this paper, we propose an efficient way of automatically verifying the result of our modeling tool, the design, for compatibility with the meta-model expressed in Alloy. We check that each instance of the model elements and their relations that is created in the design fulfills the constraints of the meta-model in Alloy. This can be efficiently done using CrocoPat[2], a tool for relational programming [8]. First, we export from our modeling tool all model element instances and their relations into a fact base as text file in the relational standard format RSF [9]. Second, we translate the constraints from the meta-model in Alloy into a relational program in RML, CrocoPat's programming language. The generated RML program contains for each constraint in the meta-model the corresponding statements to check that the constraint holds for the whole input fact base (given as RSF file to CrocoPat). If the exported fact base successfully passes the RML program, we are guaranteed that the design as produced by our modeling tool does not break any of the constraints of the meta-model in Alloy.

We proceed with explaining the translation from the meta-model in Alloy to the relational program in CrocoPat in Sect. 2. We give an overview of the functionality of our modeling tool and explain how the design it is managing can be verified in Sect. 3. In Sect. 04, we explain the limitations of our approach and how we plan to address these issues.

2. FROM ALLOY TO CROCOPAT

CrocoPat is an interpreter for imperative, relational programs. By storing the relations internally using a highly tuned representation that is based on binary decision diagrams (BDD), the tool is able to perform complex operations on large relations efficiently. To illustrate the translation from Alloy to CrocoPat, we present an Alloy meta-model that defines a simplified version of a working object. A working object, which represents a system, is a model element that is defined in our meta-model. A working object can have a number of child objects and optionally a parent object.

[1] http://alloy.mit.edu/

[2] http://www.cs.sfu.ca/~dbeyer/CrocoPat/

Figure 1 shows the corresponding Alloy code. The fields declared within the signature specify the child and parent objects. The cardinalities `set` and `lone` indicate that an object can have multiple child objects and at most one parent object. The Alloy fact `acyclic` ensures that the containment hierarchy is free of cycles, i.e., it specifies the following constraints for a working object *e*: 1) all children of *e* must have the parent object set to *e*, 2) *e* must not transitively contain itself as child, and 3) *e* must not be transitively parent of itself. Using the Alloy Analyzer we can test the consistency of this meta-model by generating some samples.

```
sig WorkingObject {
  containment: set WorkingObject,
  parent: lone WorkingObject }
fact acyclic {
all e:WorkingObject | all c:e.containment |
   (c.parent = e) and
   (e not in e.^containment) and
   (e not in e.^parent)
}
```

Fig. 1: Alloy code that specifies objects and their relationships

Figure 2 shows the RML (Relational Manipulation Language) code for CrocoPat that is semantically equivalent to the Alloy fact `acyclic` in Fig. 1. The Alloy signature `WorkingObject` is represented by a unary relation (i.e., a set) with the same name; the Alloy fields are represented by binary relations with the same name. The transitive closures are explicitly stored in our example, by the binary relations `TCContainment` and `TCParent`. The fact `acyclic` is represented by an expression whose result is assigned to relation `notAcyclic` (“FA” stands for “for all” in RML). In our relational program, we use the negation of the fact, because we are interested in providing the user with a counterexample if the fact is not valid. Therefore, the RML expression computes the set of objects that *do not* fulfill the constraints of the fact `acyclic`. The print statements output the computation results to the standard output. Note that Fig. 2 shows only the translation of the fact `acyclic`. For encoding the cardinality constraint `lone` for the field parent, we would have to add the following contraint to the conjunction in Fig. 2:

`FA(x,y, (parent(e,x) & parent(e,y)) -> x=y).`

```
TCContainment(x,z) := TC(containment(x,z));
TCParent(x,z)      := TC(parent(x,z));
notAcyclic(e)      :=
 !(WorkingObject(e) ->
   (  FA(c, containment(e,c) -> parent(c,e))
    & !TCContainment(e,e)
    & !TCParent(e,e)
   )
  );
IF( notAcyclic(e) = FALSE(x) ) {
   PRINT "Fact is valid.", ENDL;
} ELSE {
   PRINT "Fact is not valid for the
          following objects:", ENDL;
   PRINT notAcyclic(e);
}
```

Fig. 2: RML program for the Alloy code in Fig. 1

Figure 3 shows a correct fact base, i.e., relational representation of a design in RSF, for the RML program in Fig. 2. The design consists of four objects: `object0` has no parent but one child, `object1` has `object0` as parent and two children, `object11` and `object12`, which have both `object1` as parent. The unary relation `WorkingObject` contains all objects, while the parent and child relationships are listed by the binary relations `parent` and `containment`, respectively.

CrocoPat is a command-line tool that can be easily integrated into other tools. It gets as input the RML program from Fig. 2 and the RSF file from Fig. 3, and outputs to the validity result. In the negative case it outputs a list of counterexamples.

```
WorkingObject       object0
WorkingObject       object1
WorkingObject       object11
WorkingObject       object12
containment         object0       object1
containment         object1       object11
containment         object1       object12
parent              object1       object0
parent              object11      object1
parent              object12      object1
```

Fig. 3: Example RSF file of elements and their relations that form a valid design according to the specification described

3. THE SEAM MODELING TOOL

In this section we describe the SEAM modeling tool and explain how the design that is managed by the tool can be checked against the Alloy meta-model using CrocoPat.

3.1. Modeling Tool Overview

In SEAM, we build an enterprise model as collection of different interacting systems (e.g., markets, value networks, companies, technical systems). To illustrate this, we present a simple example of an EA model. At the top level, there is a `ProductMarket`. This `ProductMarket` is composed of a `SupplierValue-Network` (`SVN`) that serves a `Customer`. The term value network is a business term that we use to describe a group of companies that collaborate (to create value for a customer). The `SVN` is composed of three companies: `MarketingCo`, `Manufactur-ingCo`, and `ShippingCo`. The companies are made of departments; departments are made of employees and technical systems, etc. Fig. 4 informally represents this EA model.

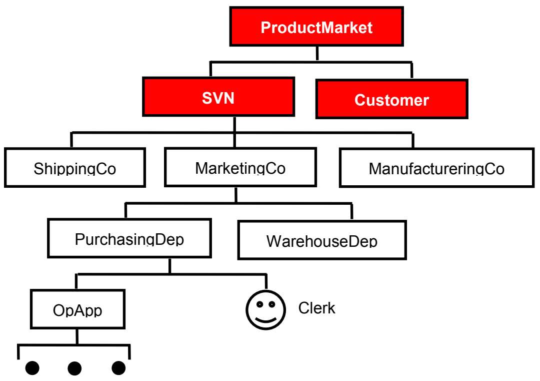
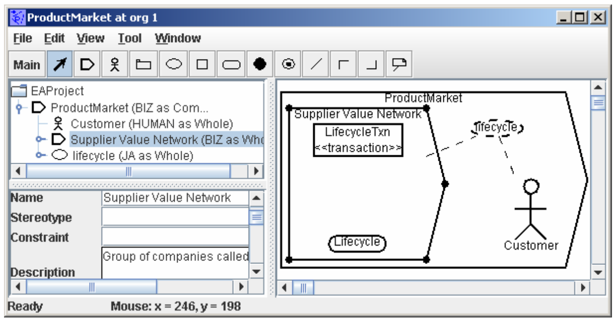


Fig. 4: Informal description of `ProductMarket`

We illustrate our modeling tool by showing two screenshots. They represent the functionality of the `SupplierValueNetwork (SVN)`. The `SVN` is considered as a black box – abstracting the companies that exist in the SVN.
Figure 5 shows an editing window of the SEAM modeling tool. To manage different parts of the same EA model, multiple editing windows can be opened at the same time. The window has three panels: The diagram panel (right side of the window) graphically shows a selected view of the EA model. The tree panel (upper left of the window) presents an overview of the EA model hierarchies: the hierarchy of working objects corresponds to the organizational hierarchy, and the hierarchy of actions and activities corresponds to the functional hierarchy. With the tree panel, the enterprise architect directs the modeling tool to generate a particular diagram from the underlying EA model. For example, in Fig. 5, `ProductMarket` is marked as a composite (cf. vertical symbol on the left side of `ProductMarket`) whereas `Customer` and `SupplierValueNetwork` are marked as non-composites (cf. horizontal symbol on the left side of their names). The resulting diagram is shown on the right of the editing window. If the architect chooses to view the `SVN` as a composite, a diagram will be shown that describes the companies that are active in the `SVN`. Or, if the architect chooses to expand the lifecycle action in the tree panel, the modeling tool shows the next functional level: `lifecycle` is then represented as a composite (i.e., an activity), and not as a whole (i.e., an action). Figure 6 shows the corresponding diagram: the behavior and properties of the `SVN` are represented (properties related pictograms are regular rectangles; behavior related pictograms are rounded rectangles).
With this tool, an enterprise architect can represent companies, together with their environment and their internal organization, which is useful for reasoning about business/IT alignment in EA projects.



Name	Supplier Value Network
Stereotype	
Constraint	
Description	Group of companies called

Fig. 5: Screenshot of our SEAM modeling tool representing a `SupplierValueNetwork` (as a whole) during the `Lifecycle` action

3.2. Meta-Model Validity Checking with Alloy

Our modeling language is defined by a meta-model expressed in Alloy. The specification has approximately 70 lines, consisting of 8 signatures and 15 facts. To validate the meta-model itself, the meta-model designer can use the Alloy Analyzer. A more elaborate approach consists in adding Alloy statements for constraint instance generation to further test the meta-model. Figure 7 shows the graphical representation (by the Alloy Analyzer) of a subset of the design shown in Fig. 5 and Fig. 6.

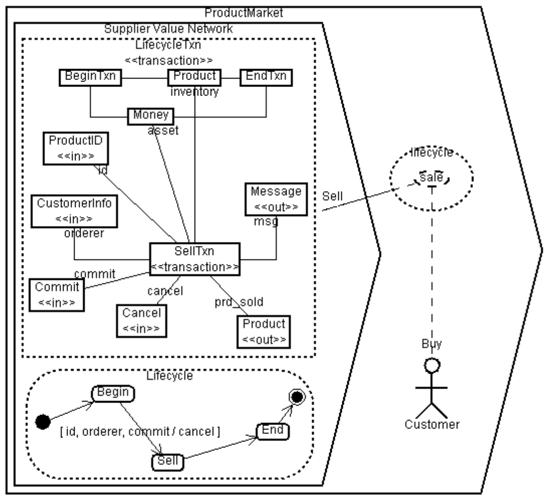


Fig. 6: `SupplierValueNetwork` (as a whole) during the `Lifecycle` activity

Once we get confidence in the quality of the Alloy meta-model, we can use the Alloy meta-model for validity checks of the design, which the user builds. To ensure the quality of the constructed design (developed and managed using the SEAM tool), we provide the capability to export a representation of the user's design into an RSF fact base. This representation can then be checked, using CrocoPat, against the Alloy meta-model. The example design discussed in the last section (Figs. 5 and 6) has a total of 300 model elements. More complicated designs are often orders of magnitude larger, in the number of elements. In the example above, the RML file had 181 lines, containing 25 relational expressions. Fig. 8 illustrates the relationships between the regular Alloy Analyzer, the Alloy/CrocoPat translator and the SEAM modeling tool. The implementation of our modeling tool follows the Model-View-Controller approach [10].

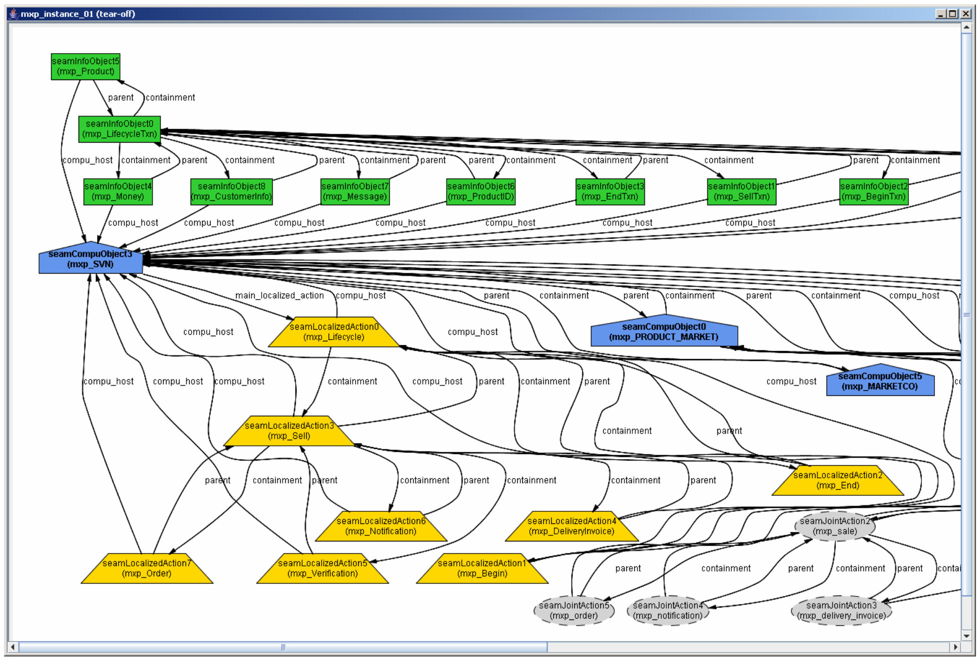

mxp_instance_01 (tear-off)
seamInfoObject5 (mxp_Product)
seamInfoObject0 (mxp_LifecycleTxn)
seamInfoObject4 (mxp_Money)
seamInfoObject8 (mxp_CustomerInfo)
seamInfoObject7 (mxp_Message)
seamInfoObject6 (mxp_ProductID)
seamInfoObject3 (mxp_EndTxn)
seamInfoObject1 (mxp_SellTxn)
seamInfoObject2 (mxp_BeginTxn)
seamCompuObject3 (mxp_SVN)
seamLocalizedAction0 (mxp_Lifecycle)
seamCompuObject0 (mxp_PRODUCT_MARKET)
seamCompuObject5 (mxp_MARKETCO)
seamLocalizedAction3 (mxp_Sell)
seamLocalizedAction2 (mxp_End)
seamJointAction2 (mxp_sale)
seamLocalizedAction6 (mxp_Notification)
seamLocalizedAction4 (mxp_DeliveryInvoice)
seamLocalizedAction7 (mxp_Order)
seamLocalizedAction5 (mxp_Verification)
seamLocalizedAction1 (mxp_Begin)
seamJointAction5 (mxp_order)
seamJointAction4 (mxp_notification)
seamJointAction3 (mxp_delivery_invoice)
parent
containment
compu_host
main_localized_action

Fig. 7: Visualization of the SVN using the Alloy Analyzer

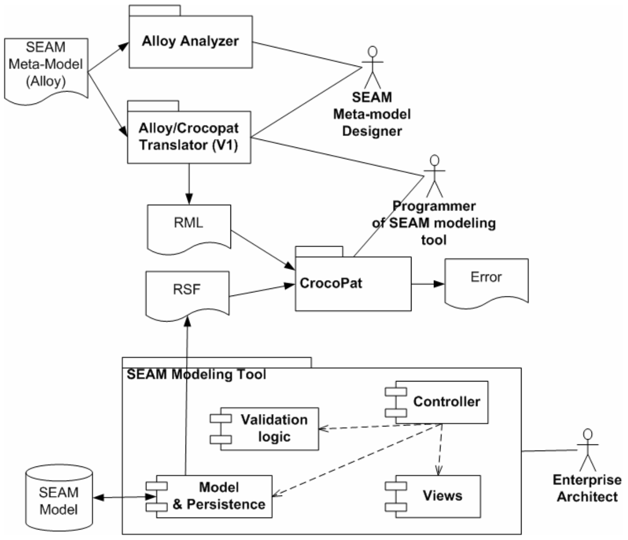


Fig. 8. The Alloy Analyzer and CrocoPat are used for checking the validity of a SEAM EA model

The Alloy/CrocoPat translator is implemented by integrating the RML code generation into the Alloy parser, using the visitor design pattern [10]. The Java class that translates Alloy to RML is approximately 1400 lines of Java source code. The translation from Alloy to CrocoPat is straightforward for most expressions (both are based on first-order predicate calculus), at the exception of some "syntactic sugar" notations in Alloy. As CrocoPat's language is based on pure predicate calculus, "abbreviations" in Alloy need to be transformed into compound expressions, i.e., the definition of the notation is inlined. Table 1 summarizes the basic Alloy operators that our prototype implementation can currently translate to RML code.

Table 1. Translation from Alloy to RML (negation for counterexample extraction included)

Alloy	CrocoPat
Quantified formulas	
`**all** x: sig_A | formula`	`CE(x) := !(sig_A(x)-> <rel expression>)`
`**no** x: sig_A | formula`	`CE(x) := sig_A(x) -> <rel expression>`
Element formulas	
`y in x.attr`	`attr(x, y)`
`x.attr != y.attr`	`!FA(z, attr(x, z) <-> attr(y, z) )`
Boolean expressions	
`left_formula && right_formula`	`<expression 1> & <expression 2>`
`left_formula || right_formula`	`<expression 1> | <expression 2>`
`left_formula <=> right_formula`	`<expression 1> <-> <expression 2>`
Relational expressions	
`left_expr & right_expr`	`<rel expression 1> & <rel expression 2>`
`left_expr | right_expr`	`<rel expression 1> | <rel expression 2>`
`left_expr - right_expr`	`<rel expression 1> & !<rel expression 2>`
Transitive closure	
`y in x.^attr`	`TC(attr(x, y)`

4. APPLICABILITY AND FUTURE WORK

The proposed approach enables us to ensure that the design produced and managed with our tool complies with the meta-model defined in Alloy. However, we have not yet proved that the tool has implemented a business logic that *prevents* the enterprise architect to develop an invalid design (i.e., during editing).
To address this issue, we are developing a more sophisticated version of the Alloy/CrocoPat translator. The idea is illustrated in Fig. 9. The goal is to directly generate code from the Alloy meta-model that is necessary to parameterize the SEAM modeling tool. The translator would then generate three kinds of files: (1) An RML program that is necessary for validity checking using CrocoPat. (2) The parameterization of the Java source files that correspond to the Alloy declarations. These Java files can be used directly in the implementation of the SEAM modeling tool as data structures that describe the EA model. (3) An XML schema that is useful to implement the persistence of data. It describes the format of the design when it is serialized.

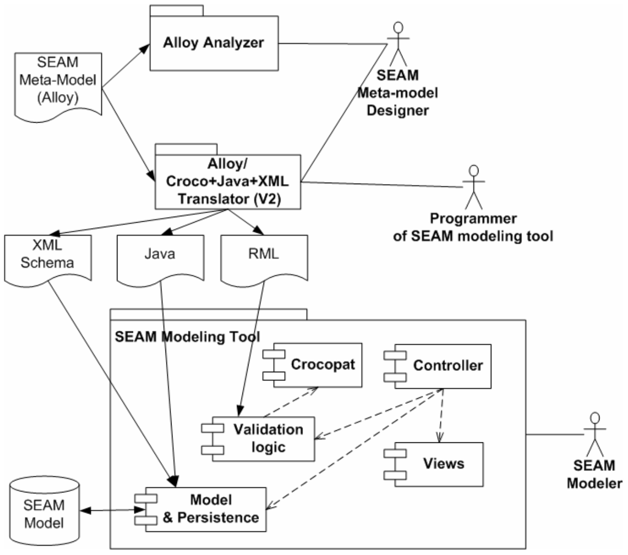


Fig. 9: Generation of the implementation code from the Alloy meta-model

In this new implementation, the validity check is continuously performed for each user command on-the-fly. User actions modify the design, and the design is instantly checked using CrocoPat. If the user performs an action that invalidates the design, then CrocoPat detects the mistake, the user action is automatically rolled back, and an error message is displayed. With such an implementation, the translation from the Alloy meta-model to the tool implementation is mechanical and guarantees a stronger relation between the modeling tool implementation and the Alloy meta-model. Another benefit is the modularity: it is much easier to test different meta-models and the modeling tool becomes parametric.

5. RELATED WORK

Since UML and OCL are becoming more and more popular, several tools specifically provide validity checks for system states by evaluating OCL invariants on object diagrams. Roclet is a modeling tool that allows the modeler not only to visually build UML class diagrams and enter OCL code but also to evaluate the entered OCL invariants in a given snapshot that is given as UML object diagram [11]. The tool highlights OCL constraints that are violated by the given snapshot. USE is another tool that can check the validation of an object diagram against some specific OCL invariants [12]. In this tool, both UML classes and OCL invariants are entered in a textual form.
In our work, we use Alloy as the specification language and CrocoPat as the validity checking engine. We are currently integrating this combination into the SEAM modeling tool.

6. CONCLUSION

Our goal is the development of a sophisticated modeling tool that produces only verified designs. Alloy is our language of choice to describe the modeling ontology (the meta-model). CrocoPat is our verifier of choice to efficiently verify the design against its specification. We have shown in this paper that constraints formulated in Alloy can easily be translated into CrocoPat programs. This new feature ensures that both the user and our implementation of the business logic in the modeling tool do not introduce inconsistencies into the design. We outlined how we will achieve an implementation of our modeling tool that prevents the user from constructing invalid designs.

7. REFERENCES

[1] J. Luftman and E.R. McLean. Key Issues for IT Executives. *MIS Quarterly Executive*, vol. 3, 2004.
[2] A. Wegmann. On the Systemic Enterprise Architecture Methodology (SEAM). In *Proc. ICEIS*, Angers, France, 2003.
[3] L.S. Lê and A. Wegmann. SeamCAD: Object-Oriented Modeling Tool for Hierarchical Systems in Enterprise Architecture. In *Proc. HICSS*, Track 8, p.179c, IEEE, 2006.
[4] OMG. ISO/IEC 10746-1, 2, 3, 4 ITU-T Recommendation, X.901, X.902, X.903, X.904, Reference Model of Open Distributed Processing. 1995-1996.
[5] D. Jackson. Alloy: A lightweight object modelling notation. *ACM Trans. Softw. Eng. Methodol.*, **11**(2):256–290, 2002.
[6] L.S. Lê and A. Wegmann. An RM-ODP Based Ontology and a CAD Tool for Modeling Hierarchical Systems in Enterprise Architecture. In *Proc. WODPEC*, Enschede, NL, 2005.
[7] L.S. Lê and A. Wegmann. Definition of an Object-Oriented Modeling Language for Enterprise Architecture. In *Proc. HICSS*, Track 8, p.222a, IEEE, 2005.
[8] D. Beyer. Relational Programming with CrocoPat. In *Proc. ICSE*, pages 807-810, ACM, 2006.
[9] D. Beyer and A. Noack. CrocoPat 2.1 Introduction and Reference Manual. Tech. rep. UCB//CSD-04-1338, Computer Science Division (EECS), University of California, Berkeley, 2004.
[10] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns*. Addison-Wesley, 1995.
[11] S. Marković and T. Baar. An OCL Semantics Specified with QVT. In *Proc. MoDELS*, LNCS 4199, pages 661-676, Springer, 2006.
[12] M. Richters, M. Gogolla. Validating UML Models and OCL Constraints. In *Proc. UML*, pages 265-277, Springer, 2000.