Enhancing UML Models: A Domain Analysis Approach

Iris Reinhartz-Berger

Address: Department of Management Information Systems,

University of Haifa, Haifa 31905, Israel

Tel: 972-4-8288502

Fax: 972-4-8288522

e-mail: iris@mis.haifa.ac.il

Arnon Sturm

Address: Department of Information Systems Engineering,

Ben-Gurion University of the Negev, Beer Sheva 84105, Israel

Tel: 972-8-6479337

Fax: 972-8-6479337

e-mail: sturm@bgu.ac.il

Enhancing UML Models: A Domain Analysis Approach

Abstract. UML has been largely adopted as a standard modeling language. The emergence of UML from different modeling languages that refer to various system aspects causes a wide variety of completeness and correctness problems in UML models. Several methods have been proposed for dealing with correctness issues, mainly providing internal consistency rules but ignoring correctness and completeness with respect to the system requirements and the domain constraints. In this paper, we propose addressing both completeness and correctness problems of UML models by adopting a domain analysis approach called Application-based DOmain Modeling (ADOM). We present experimental results from our study which checks the quality of application models when utilizing ADOM on UML. The results advocate that the availability of the domain model helps achieve more complete models without reducing the comprehension of these models.

Keywords: UML, Domain Analysis, Model Correctness, Model Completeness

INTRODUCTION

Conceptual modeling is fundamental to any area where one has to cope with complex real world systems. The most popular, de-facto modeling language today is UML, which is used for specifying, visualizing, constructing, and documenting the artifacts of software systems, as well as for business modeling and other non-software systems (OMG-UML, 2003; OMG-UML, 2006). Although UML provides convenient, standard mechanisms for software engineers to represent high-level system designs as well as low-level implementation details (Tilley & Huang, 2003), it also introduces a variety of correctness and completeness problems. According to Major and McGregor (1999), *correctness* is measured as how accurately the model represents the information specified within the requirements. For defining the correctness of a model, a source that is assumed to be (nearly) infallible is identified. This source, termed a “test oracle”, is usually a human expert whose personal knowledge is judged to be sufficiently reliable to be used as a reference. The accuracy of the model representation is measured relatively to the results expected by the oracle. *Completeness*, on the other hand, deals with the necessity and usefulness of the model to represent the real life application, as well as the lack of required elements within the model (Major & McGregor, 1999). In other words, completeness is judged as to whether the information being modeled is described in sufficient details for the established goals. This judgment is based on the model’s ability to represent the required situations, as well as on the knowledge of experts.

Different studies concluded that it is difficult to model a correct and consistent application using UML and even to understand such a specification (Kabeli & Shoval, 2001; Peleg & Dori, 2000; Dori, 2001; Reinhartz-Berger & Dori, 2005; Siau & Cao; 2001). Several methods have been suggested for checking the correctness of UML models. However, these mainly deal with syntactic issues directly derived from the modeling language metamodel, neglecting the correctness and completeness of the models with respect to the domain constraints and the system requirements.

In this research we utilize the Application-based DOmain Modeling (ADOM) approach (Reinhartz-Berger & Sturm, 2004; Sturm & Reinhartz-Berger, 2004), whose roots are in the area of domain engineering, for enhancing UML models. ADOM enables specifying and modeling domain artifacts that capture the common knowledge and the allowed variability in specific areas, guiding the development of particular applications in the area, and validating the correctness and completeness of applications with respect to their relevant domains. ADOM does these with regular application and software engineering techniques and languages, bridging the gap between the different abstraction levels at which application and domain models reside and reducing learning and training times. We present initial results from our study which checks the comprehension and quality of UML models when applying ADOM.

Following the introduction we review relevant works from related areas and briefly introduce the ADOM approach, emphasizing its usage for developing correct and complete UML models. We then elaborate on the experiment we conducted, its hypotheses, settings, and results. Finally, we summarize the advantages and limitations of the proposed approach, raising topics for future research.

LITERATURE REVIEW

Shull et al. (2000) defined six types of software defects that can be found in object-oriented designs: missing information, incorrect facts, inconsistent information, ambiguous information, extraneous information, and miscellaneous defects. Incorrect facts, inconsistent information, ambiguous information, and extraneous information refer to the model correctness, while missing information refers to completeness.

Several solutions have been proposed over the years for handling these defects, mainly concerning consistency and integration problems. These solutions can be roughly divided into translation and verification approaches. Translation approaches, such as Bowman et al. (2002), Rasch & Wehrheim (2002), Mens et al. (2003), Große-Rhode (2001), and Baresi & Pezze (2001), translate multi-view models into formal languages that can be analyzed by model checkers. After detecting inconsistencies or mistakes a backward process should be applied, translating the locations where the defects were found back to the multi-view models in order to enable the developers to fix them. Whittle (2000) surveyed some of the attempts to formalize the semantics of UML by applying formal methods for analyzing UML models. His main conclusion was that UML semantics is largely informal and, hence, more effort should be directed towards making the semantics precise. Verification approaches, on the other hand, such as Chiorean et al. (2003), Bodeveix et al. (2002), Engels et al. (2002), and Nentwich et al. (2003), present testing or validation algorithms which check inconsistencies and contradictions between various views. They require sophisticated environments which include test drivers, interpreters, controllers, etc. Reinhartz-Berger (2005) suggests a top-level approach that glues the different UML views into one coherent system throughout the entire development process life-cycle. However, all these works refer to the syntax of the models only. Moreover, none of them deals with completeness issues and errors that originate from the constraints imposed by the application domain.

Examining common knowledge and utilizing it for developing applications may help construct better applications in less time and efforts. Indeed, pattern-based modeling approaches, such as Neal and Linington (2001) and Mapelsden et al. (2002), aim at helping produce better design and implementation of applications by reusing solutions for recurring design problems. However, these are usually too abstract to be used directly, refer mainly to the common features of the solutions in the different contexts, and require further expertise in order to correctly apply the patterns.

Domain analysis refers to the commonality and variability of sets or families of applications, defined as domains (Valerio et al., 1997). It specifies the basic elements of the domain, organizes an understanding of the relationships among these elements, and represents this understanding in a useful way (Champeaux et al., 1993). Three main groups of domain analysis techniques are architectural-based, feature-oriented, and metamodeling. Architectural-based methods (e.g., Neighbors, 1989; Meekel et al., 1997) define the domain knowledge in components, libraries, or architectures, which may be reused in an application as they are, but can be also modified to support the particular requirements at hand. Feature-oriented methods (e.g., Gomaa & Kerschberg, 1995; Kang et al., 1990; Kang et al., 1998; Gomaa, 2004) suggest that a system specification will be derived by tailoring the domain model according to the features desired in a specific system. Metamodeling techniques (e.g., Schleicher & Westfechtel, 2001; Gomaa and Eonsuk-Shin, 2002; Nordstrom et al., 1999) enable definition of domains as metamodels that serve both for capturing domain knowledge and validating particular applications in the domain.

Similarly to the software engineering field, the area of business process design and management also promote domain analysis in the form of reference models, which are models used for supporting the construction of other models. Reference models were originally suggested as a vehicle for enhancing the development of information systems (Schuette & Rotthowe, 1998; Fettke & Loos, 2003; Thomas, 2005), but they also provide generic knowledge about business processes in order to assist in their design in specific enterprises.

We decided to use a specific domain analysis approach, called Application-based DOmain Modeling (ADOM), which can serve as a method for guiding and validating the development of more complete and correct application models in a specific domain. ADOM has already presented in (Reinhartz-Berger & Sturm, 2004; Sturm & Reinhartz-Berger, 2004; Reinhartz-Berger et al., 2005; and Soffer et al., 2007). In this research, we focus on the ability of ADOM to enhance the correctness and completeness of UML models in given domains. Furthermore, we provide empirical evidence for the support of ADOM for these issues, somehow justifying the costs and efforts required for developing domain models.

THE APPLICATION-BASED DOMAIN MODELING (ADOM) APPROACH

The ADOM approach is based on a three layered architecture: application, domain, and (modeling) language. Being influenced from the classical framework for metamodeling presented in OMG-MOF (2003), the application layer, which is equivalent to the model layer (M1), consists of models of particular applications, including their structure (scheme) and behavior. The language layer, which is equivalent to the metamodel layer (M2), includes metamodels of modeling languages. The intermediate domain layer consists of specifications of various domains, such as web applications, multi agent systems, and process control systems.

ADOM is a general approach which can be used in conjunction with different modeling languages, but when adopting ADOM with a specific modeling language, this language is used for both application and domain layers, easing the task of application design and validation by employing the same terminology in both layers. The only requirement from the modeling language used in conjunction with ADOM is that it will have a classification mechanism that enables categorizing groups of elements. The stereotype and profile mechanisms in UML are examples of such mechanism.

A domain model in ADOM captures generic knowledge (know-how), in terms of common elements and the allowed variability among them. In particular, the classification mechanism is used in the domain layer in order to denote the multiplicity indicators of the different domain model elements, where a multiplicity indicator specifies a range for the number of specializations of a specific domain element that may be included in an application model in that domain.

An application model can be constructed on the basis of the knowledge captured in the domain model. In this case, we refer to the application model as an *instantiation* of the domain model. Instantiation can be mainly achieved by configuration or specialization operations, performed at design time (when the application model is created). *Configuration* is the selection of a subset of existing elements from a domain model for the purpose of specifying a lawful specific application model. *Specialization*, on the other hand, is the result of concretization of a domain model element into a specific application model element. The generic (domain) elements can be specialized through operations of refinement, sub-typing, and contextual adoption, so that one generic element may be specialized into more than one element in the specific application model (Soffer et al., 2007). The relations between a generic element and its instantiations are maintained by the classification mechanism of the modeling language. In addition, some generic elements may be omitted and some new specific elements may be inserted to the specific (application) model. Nevertheless, the domain knowledge embedded in the generic model must be maintained in the specific one.

The Theoretical Foundations of ADOM

ADOM advocates the application of two main cognitive theory principles: analogical problem solving and analogical reasoning. These principles promote the usage of existing knowledge for solving new problems. Analogy was found as a powerful tool for understanding new situations and finding appropriate solutions for them (Yanowitz, 2001). From the early 90's, its possible usage for requirements analysis was introduced and discussed (e.g., Maiden & Sutcliffe, 1992). According to Holyoak (1984), analogical problem solving is performed in four steps: (1) forming a mental representation of both the reference and target, (2) generating the relevant analogy, (3) mapping across the features, and (4) generating the solution based on the analogy. In ADOM, the domain model is constructed according to the knowledge gained with previous applications and literature review. Having a domain model, it is used as a reference for developing the target application model. Both generating the analogy and mapping across features are done by the customization and specialization operations and the classification mechanism of the used modeling language. Finally, the ADOM approach supports additions to the application models in order to generate complete solutions that are based on the analogy.

In this paper we use ADOM-UML, in which UML version 1.5 is used as the underlying modeling language for specifying both applications and domains and the analogy between them. We chose ADOM for enhancing UML models because of the following main reasons. First, ADOM treats domains similarly to applications, enabling the usage of the same techniques to both application and domain levels. Hence, it is more accessible to software engineers and enables the specification of both behavioral and structural constraints. Second, ADOM supports the construction of legal application models throughout the entire development process and does not execute model checking or validation algorithms at certain development stages. Thus, it helps avoid and handle model incorrectness and incompleteness at early development stages. In what follows we elaborate on representing domain models in ADOM-UML and specifying correct and complete application models.

ADOM-UML Domain Layer

In the language layer of ADOM-UML, a new <<multiplicity>> stereotype is defined in order to represent the multiplicity indicators. We use the stereotype mechanism for defining multiplicity constraints since UML multiplicity mechanism is applicable only for associations and attributes and we wish to express multiplicity constraints on all model elements. Furthermore, we apply the stereotype mechanism also in cases where the multiplicity mechanism is applicable in order to preserve uniformity and avoid confusion between the meaning of multiplicity in the application and domain models.

The <<multiplicity>> stereotype has two associated tagged values, min and max, which define the lowest and upper most multiplicity boundaries, respectively. For simplicity purposes, we shorten the notation <<multiplicity min=m max=n>> to <<m..n>> and define <<1..1>> as the default (i.e., this stereotype may not explicitly appear). These multiplicity stereotypes are used in the domain layer, where the main concepts of the domain and the relations among them are specified. This type of stereotypes constrains the number of the specializations of a domain model element in the application models to be built. This way a variety of correctness and completeness rules (constraints) can be defined at the domain level, enforcing their validation in all applications in the domain. Examples of such rules, which are specified in the process control systems (PCS) domain model that appears in Appendix A, are given below. Note that applications in the PCS domain monitor and control the values of certain variables through a set of components that work together to achieve a common objective or purpose (Duffy, 2004). However, as will be demonstrated latter, their purposes and implementations may be quite different.

Rule 1 (from the UC diagram): An application in the PCS domain interacts with three types of actors, **Operator**, **Sensor**, and **Controlled Device**, each of which must be specialized at least once in any application in this domain.

Rule 2 (from the UC diagram): Each application in the domain has at least one use case in the following categories: **System Settings**, **System Activation**, **Monitoring & Acting**, and **Checking**.

Rule 3 (from the class diagram): Each application in the domain has exactly one class classified as **Controller** and at least one class in each of the following categories: **SensorInfo**, **ControlledDeviceInfo**, **ControlledElement**, and **ControlledValue**.

Note that the domain model also provides additional knowledge on the structure of each concept, including its attributes, operations, and relations to other concepts. Each **ControlledElement** class, for example, has at least one attribute classified as **controlledElementIdentity**, at least one operation classified as **monitorAndAct,** and at least one operation classified as **checkCondition** (each of which returns a Boolean value). In addition, **ControlledElement** may have enumerated attributes classified as **controlledElementStatus**.

Rule 4 (from the sequence diagram): Each application in the domain deals with monitoring and acting in the following way. The **Controller** activates (in a loop) a

monitorAndAct operation on the **ControlledElements**. This operation acts in two stages: in the first stage the condition is checked, while in the second stage the action takes place. The activation part of the sequence is embedded within the condition checking part and each one of them can be repeated several times.

Rule 5 (from the statechart diagram): Each **ControlledDevice** has exactly one "**off**" state and at least one "**on**" state. The transition between "**off**" and "**on**" states is done by an **action**, while no additional information is provided in the domain level about the transitions between "**on**" and "**off**" states.

3.3 ADOM-UML Application Layer

An application model may use a domain model as guidelines (or analogy) for creation and as a validation template for checking that all the constraints enforced by the domain model are actually fulfilled by the application at hand. For these purposes, elements in the application model are classified according to the elements declared in the domain model using the UML stereotype mechanism. A model element in an application model is required to preserve the constraints of its stereotypes in the relevant domain model.

Returning to our PCS example, we describe in this section two applications in the domain: a Home Climate Control (HCC) application and a Water Level Control (WLC) system. The HCC application ensures that the temperature in the rooms of a house remains in the closed range [TL, TH] and the humidity in these rooms remains in the closed range [HL, HH]. Each room has its own limit values (TL, TH, HH, and HL) which are configurable. The actual levels of temperature and humidity are measured by thermometers and humidity gauges, respectively. An air conditioner and a water sprayer are installed in each room, enabling changing the temperature and humidity at will. The ADOM-UML model of the HCC application appears in Appendix B.

The purpose of the WLC application is to monitor and control the water levels in tanks, ensuring that the actual water level is always in the closed range [Lowest Limit, Highest Limit]. The values of the lowest and highest limits are configurable. The actual level is measured by a boundary stick. The tank is also coupled to emptying faucets that drain water from the tank and to filling faucets that inject water into the tank. The ADOM-UML model of the WLC application appears in Appendix C.

Although different, both applications use the knowledge captured in the PCS domain model and preserve its constraints. In particular, they both maintain the five rules exemplified in Section 3.2. Note that these rules may not explicitly appear in the requirement specification of a particular application, as they may be common property of the domain. Therefore, the designer is responsible for keeping the model complete and correct with respect to these rules. Explicitly specifying these rules in the form of domain models may contribute and help the designer to better perform his/her tasks.

THE EXPERIMENT

In order to verify the usefulness of the ADOM approach for enhancing UML application models, we conducted an experiment, whose hypotheses, settings, and results are reported below.

Experiment Hypotheses

In the experiment we aimed at checking the following three hypotheses.

Hypothesis #1: **Application models are more completely developed when a domain model is available.** This hypothesis is derived from the observation that domain models may include relevant elements and constraints that do not explicitly appear in the requirements of each application in the domain. Furthermore, "best practices" can be incorporated into the domain models as optional elements (i.e., elements whose minimal multiplicity indicator is 0), helping the designer not to miss information.

Hypothesis #2: **Application models are more correctly developed when a domain model is available.** Here, again, wrong interpretation of requirements may be avoided by the domain artifacts and knowledge.

Hypothesis #3: **The comprehension of application models remains unchanged when the relevant domain model and elements are added.** The reason for this hypothesis originates from the observation that domain and application models belong to two different abstraction levels. When answering concrete questions about the applications, the more abstract domain elements might generalize the needed information, blurring the sought answer. However, the existence of these domain elements may help answer questions which relate to generalized application information.

Experiment Settings

The subjects of the experiment were third year students in a four-year engineering B.Sc. program at Ben-Gurion University of the Negev, Israel, who took the course “Object-Oriented Analysis and Design” at the winter semester of the 2004-5 academic year. All of them were students of the Information Systems Engineering program and had no previous knowledge or experience in system modeling and specification. During the course, the students studied mainly UML and its applicability to software analysis and design, while the last lecture was devoted to ADOM.

The experiment took place during the final three-hour examination of the course. The examination contained two tasks, one of which was related to the reported experiment. In this task the students were asked to respond to nine true/false comprehension questions about the HCC application and to build a model of a WLC application. The students were told that both applications belong to the same PCS domain. The comprehension questions are listed along with their expected answers in Appendix D, which also includes the modeling question that refers to the WLC application. An acceptable model to this application is given in Appendix C.

The students were divided arbitrarily into two groups of 34 and 36 students. Each group got a different test form type, ADOM-UML and "regular" UML, respectively. The "regular" UML test form included a UML model of the HCC application, as given in Appendix B without the stereotypes. The ADOM-UML test form included the PCS domain model and the HCC application model as given in Appendices A and B, respectively. The students were provided with alternating form types according to their seating positions, so this arbitrary division into the two experimental groups closely approximated random division. Executing a t-test on the average grades of the students in their studies, we indeed found that no significant difference exists between the two groups ($t = 0.32$, $p \sim 0.75$).

In order to validate the correctness and completeness of the models that participate in the experiment, as well as to check that the comprehension questions can be accurately answered and the WLC application can be accurately modeled in both form types, four UML design experts examined them carefully. Only after reaching an agreement on all the aforementioned issues, the experiment was conducted.

We also addressed ethical concerns that may rise using the author’s students as participants (Singer & Vinson, 2002). In particular, the students were notified at the beginning of the semester about the exam being used as an experiment; the students had the opportunity of getting a grade in the course without participating in the experiment (by taking term B of the exam); the grades of the two test forms were normalized; and confidentiality was kept throughout the entire data grading and analysis processes, so no identification of the subjects can be done.

Experiment Results

The comprehension and modeling questions were checked according to a pre-defined detailed grading policy, which included potential errors along with the number of points that should be reduced in case of error occurrences. Each comprehension question could score a maximum of 2 points (18 points in total), while the modeling question could score as much as 32 points. Incomplete answers, or incorrect answers, scored less according to the detailed grading policy.

Table 1 summarizes the average scores of the comprehension, modeling, and overall grades. A t-test, which was used to analyze these results, showed that although the average comprehension score of the ADOM-UML group was higher than that of the "regular" UML group, it was not found as statistically significant ($p<0.094$). This outcome can be considered as in-line with our third hypothesis, as we claimed that domain models sometimes help find generalized answers (i.e., answers that are relevant to several applications in the domain or to several specializations of the same domain element) and sometimes blur the sought answers (for questions that are individual to the specific application). Since the questions in the experiment belong to both categories, no significant differences were found.

Table 1. Results of the comprehension, modeling, and overall grades

	Average score		**t**	**p-value**
	ADOM-UML	**Regular UML**		
Comprehension	**76.31%**	68.98%	1.698	<0.094
Modeling	**89.11%**	77.73%	3.605	**<0.001**
Overall	**84.50%**	74.58%	3.214	**<0.002**

However, the statistical analysis shows that the availability of the domain model was very important for modeling a new application in the domain. This is especially true, since the students that participated in the experiment were non-experts (in the domain, the modeling language, and the development task).

In order to carry out an in-depth analysis of the domain model influence on both correctness and completeness of application models, we further checked the average amount of points reduced due to incompleteness and incorrectness, whereas incompleteness referred to missing elements and correctness was measured in terms of redundancy, incorrect facts, and inconsistency among the diagrams. Note that some of the points were reduced due to miscellaneous defects (Shull et al., 2000) and, thus, they are omitted from the calculations regarding correctness and completeness.

Table 2 presents the average amount of points reduced due to incompleteness and incorrectness. As claimed in hypothesis #1, the results clearly show that ADOM-UML helped gain a more complete model. We believe that the main reason for this outcome is using the ADOM-UML domain model as guidelines (or an analogy) for building the application model, rather than starting from scratch or from a similar application. These guidelines were applied properly as the students had another application from the same domain (the HCC model) that applied the same guidelines.

Regarding correctness, Table 2 shows that the students had fewer errors when using the ADOM-UML. However, this was not found as statistically significant ($p<0.062$). The main differences between the two groups were that students who used the "regular" UML test forms had significantly less errors related to the class diagram, while students who used the ADOM-UML test forms had significantly less errors related to messages in the sequence diagram and to states and their transitions in the state diagram. We believe that the reason for the statistical insignificance of these results was the similarity between the two applications: the given HCC model and the requested WLC model. Our belief relies on the observation that the students who got "regular" UML test forms consider the HCC model as a reference for modeling the WLC system. This could be done since the particular applications are very similar. However, in the general case, only the domain model can serve as a template to guide the developer in the development of new domain-specific applications.

Table 2. The average percentages of points reduced due to completeness and correctness errors

Inspected aspect	ADOM-UML	Regular UML	t	p-value
Completeness	**5.59%**	10.84%	-3.324	**<0.002**
Correctness	**5.34%**	7.66&	-1.904	<0.062

SUMMARY AND FUTURE WORK

In this paper we suggest utilizing domain models for improving application model quality in terms of correctness and completeness. We use a specific domain analysis method, called Application-based DOmain Modeling (ADOM), that enables specifying both domain and application models with similar software engineering techniques and languages and creating analogy among them. When developing a particular application in the domain, the domain model is used as a reference and the application model can be validated against the relevant domain model in order to detect completeness and correctness errors.

We empirically validated our approach on undergraduate students, i.e., inexperienced users who need additional tools and techniques in order to develop qualitative application models. The results presented in this paper suggest that the availability of the domain model help develop better application models, mainly with respect to their completeness, without affecting their comprehension. When developing totally new applications inexperienced designers tend to create erroneous models, but even experienced ones cannot anticipate the implication of a change on an overall model (Sunye et al, 2001). Indeed, Lange et al. (2006) showed that model defects often remain undetected, even if practitioners check the model attentively. These results may advocate and justify the costs and efforts required in developing complete and correct domain models for mature, stable, economically viable domains, as these domain models can be used for guiding the development of high quality applications in these domains.

Similarly to other domain analysis approaches, ADOM can be criticized as inhibiting creativity, and its applicability for domains where consensus cannot be reached by domain experts may be questioned. However, from the creativity point of view, Kolle (2005) states that basing on existing solutions not necessarily decrease creativity, as it may give rise to creativity related to the specific problem at hand. Regarding scalability, domain models in ADOM capture not only the commonality, but also the variability allowed among applications in a specific domain, so that consensus does not have to be reached before modeling the domain. Furthermore, although examined on a small and relatively simple example, ADOM has the expressiveness to support the specification of more complex domain rules and constraints. Indeed, we are currently in the process of checking the expressiveness of ADOM in different modeling languages and contexts. We have already adopted ADOM for business process modeling with UML activity diagrams (Reinhartz-Berger et al., 2005) and EPC (Soffer et al., 2007), for requirement and design modeling with OPM (Sturm et al., 2006), and for web site development with Tersus (Tersus, 2006).

In the future, we plan to continuously evaluate the effectiveness of using ADOM (by novice and experienced designers) to support the construction of consistent, correct, and complete application models in various modeling languages. In addition, we work on developing a formal and accurate process for instantiating and utilizing domain models. We believe that this process, which will be supported by a CASE tool, will help improve the accuracy, correctness, and completeness of the resultant application models.

REFERENCES

Baresi, L. & Pezze, M. (2001). On Formalizing UML with High-Level Petri Nets. Concurrent Object-Oriented Programming and Petri Nets, 276-304.

Bodeveix, J.P., Millan, T., Percebois, C., Le Camus, C., Bazex, P., Feraud, L. & Sobek, R. (2002). Extending OCL for Verifying UML Models Consistency, Workshop on Consistency Problems in UML-based Software Development, 5th International Conference on the Unified Modeling Language- the Language and its applications (UML'2002), Dresden, Germany, 75-90.

Bowman, H., Steen, M., Boiten, E.A. & Derrick, J. (2002). A Formal Framework for Viewpoint Consistency, Formal Methods in System Design, 21 (2), 111-166.

Champeaux, D. de, Lea, D. & Faure, P. (1993). Object-Oriented System Development, Addison Wesley.

Chiorean, D., Pasca, M., Carcu, A., Botiza, C. & Moldovan, S. (2003). Ensuring UML models consistency using the OCL Environment. Workshop on OCL 2.0 - Industry standard or scientific playground?, 6th International Conference on the Unified Modeling Language - the Language and its applications (UML'2003), San Francisco (2003).

Dori, D. (2001). Object-process methodology applied to modeling credit card transactions. Journal of Database Management, 12 (1), 4-14.

Duffy, D. J. (2004). Domain Architectures: Models and Architectures for UML Applications. John Wiley & Sons.

Engels, G., Hausmann, J.H., Heckel, R. & Sauer, S. (2002). Testing the Consistency of Dynamic UML Diagrams. Proc. 6th International Conference on Integrated Design and Process Technology (IDPT 2002), Pasadena CA, http://www.uni-paderborn.de/cs/ag-engels/Papers/2002/EngelsHHS-IDPT02.pdf.

Fettke, P. & Loos, P. (2003). Classification of Reference Models - A Methodology and its Application, Information Systems and e-Business Management, 1 (1), 35-53.

Gomaa, E. & Kerschberg, L. (1995). Domain Modeling for Software Reuse and Evolution, Proceedings of Computer Assisted Software Engineering Workshop (CASE 95).

Gomaa, H. (2004). Designing Software Product Lines with UML: From Use Cases to Pattern-based Software Architectures, The Addison-Wesley Object Technology Series.

Gomaa, H. & Eonsuk-Shin, M. (2002). Multiple-View Meta-Modeling of Software Product Lines, Proceedings of the Eighth IEEE International Conference on Engineering of Complex Computer Systems.

Große-Rhode, M. (2001). Integrating Semantics for Object-Oriented System Models, 28th International Colloquium on Automata, Languages and Programming (ICALP 2001), Crete, Greece, Lecture Notes in Computer Science 2076, 40-60. http://i11www.ira.uka.de/~baar/oclworkshopUml03/papers/06_ensuring_uml_model_consistency.pdf

Holyoak, K.J. (1984). Analogical thinking and human intelligence. Advances in the Psychology of Human Intelligence. Ed: R. J. Sternberg. Hillsdale, New Jersey: Lawrence Erlbaum Associates, 199-230.

Kabeli, J. & Shoval, P. (2001). FOOM: functional- and object-oriented analysis & design of information systems - an integrated methodology, Journal of Database Management, 12 (1), 15-25.

Kang, K. C., Kim, S., Lee, J., Kim, K., Shin, E. & Huh M. (1998). FORM: A feature-oriented reuse method with domain-specific reference architectures, Annals of Software Engineering, 5 (1), 143-168.

Kang, K., Cohen, S., Hess, J., Novak, W. & Peterson, A. (1990). Feature-Oriented Domain Analysis (FODA) Feasibility Study, CMU/SEI-90-TR-021 ADA235785.

Kolle, A. G. (2005). Analogical problem-solving in design, Evaluating the feasibility and benefits of the methodical use of biological analogies in solving design problems, Technical report, http://design.ntnu.no/forskning/artikler/2005/artikkel_Audun_Kolle.pdf.

Lange, C.F.J., Chaudron, M.R.V. & Muskens J. (2006). In Practice: UML Software Architecture and Design Description. IEEE Software, 23 (2), 40-46.

Maiden, N. & Sutcliffe, A. (1992). Exploiting reusable specifications through analogy. Communication of the ACM, 35(4), 55-64.

Major, M. & McGregor, J. (1999). Using Guided Inspection to Validate UML Models, the 24th Annual IEEE/NASA Software Engineering Workshop.

Mapelsden D, Hosking, J. & Grundy, J. (2002). Design Pattern Modeling and Instantiation using DPML, TOOLS Pacific 2002, 3-11.

Meekel, J., Horton, T. B., France, R. B., Mellone, C. & Dalvi, S. (1997). From domain models to architecture frameworks, Proceedings of the 1997 symposium on Software reusability, 75-80.

Mens, T., Van Der Straeten, R. & Simmonds, J. (2003). Maintaining Consistency between UML Models Using Description Logic. Workshop on Consistency Problems in UML-based Software Development II, 6th International Conference on the Unified Modeling Language – the Language and its applications (UML'2003), San Francisco, 71-77.

Neal, S. & Linington, P. (2001), Tool Support for Development using Patterns, International Enterprise Distributed Object Computing, 237-248.

Neighbors, J. (1989). Draco: A Method for Engineering Reusable Software Systems, in T. Biggerstaff & A. Perlis. Software Reusability. Volume I: Concepts and Models. ACM Press, Frontier Series, Addison-Wesley, Reading, 295-319.

Nentwich, C., Emmerich, W., Finkelstein, A. & Ellmer, E. (2003). Flexible consistency checking, ACM Transactions on Software Engineering and Methodologies, 12 (1), 28-63.

Nordstrom, G., Sztipanovits, J., Karsai, G. & Ledeczi, A. (1999). Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments, Proceedings of the IEEE Sixth Symposium on Engineering Computer-Based Systems (ECBS), 68-74.

OMG-MOF (2003). Meta-Object Facility (MOF™), version 1.4.

OMG-UML (2003). The Unified Modeling Language (UML™), version 1.5.

OMG-UML (2006). UML 2.0 Superstructure, 2006.

Peleg, M. & Dori, D. (2000). The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods. IEEE Transaction on Software Engineering, 26 (8), 742-759.

Rasch, H. & Wehrheim, H. (2002). Consistency Between UML Classes and Associated State Machines. Workshop on Consistency Problems in UML-based Software Development, 5th International Conference on the Unified Modeling Language- the Language and its applications (UML'2002), Dresden, Germany, 46-60.

Reinhartz-Berger, I. (2005). Conceptual Modeling of Structure and Behavior with UML – The Top Level Object-Oriented Framework (TLOOF) Approach, the 24th International Conference on Conceptual Modeling (ER'2005), Lecture Notes in Computer Science 3716, 1-15.

Reinhartz-Berger, I. & Dori, D. (2005). OPM vs. UML – Experimenting Comprehension and Construction of Web Application Models. Empirical Software Engineering (EMSE), 10 (1), 57-80.

Reinhartz-Berger, I., Soffer, P., & Sturm, A. (2005). A Domain Engineering Approach to Specifying and Applying Reference Models, Proceedings of Enterprise Modeling Information Systems Architecture (EMISA'05), pp. 50-63.

Reinhartz-Berger, I. & Sturm, A. (2004). Behavioral Domain Analysis – The Application-based Domain Modeling Approach, the 7th International Conference on the Unified Modeling Language (UML'2004), Lecture Notes in Computer Science 3273, 410-424.

Schleicher, A. & Westfechtel, B. (2001). Beyond Stereotyping: Metamodeling Approaches for the UML, Proceedings of the 34th Annual Hawaii International Conference on System Sciences, 1243-1252.

Schuette, R., & Rotthowe, T. (1998). The Guidelines of Modeling – an Approach to Enhance the Quality in Information Models, Proceedings of the 17th International Conference on Conceptual Modeling (ER'98), Lecture Notes In Computer Science 1507, Springer-Verlag, Berlin, 240-254.

Shull, F., Rus, I. & Basili V, (2000). How Perspective-Based Reading Can Improve Requirements Inspections, IEEE Computer, 33 (7), 73-79.

Siau, K. & Cao, Q. (2001). Unified Modeling Language: A Complexity Analysis, Journal of Database Management, 12 (1), 26-34.

Singer, J. & Vinson, N. G., (2002). Ethical Issues in Empirical Studies of Software Engineering, IEEE Transaction on Software Engineering, 28 (12), 1171-1180.

Soffer, P., Reinhartz-Berger, I,, and Sturm, A (2007), Facilitating Reuse by Specialization of Reference Models for Business Process Design, Accepted to the 8th Workshop on Business Process Modeling, Development, and Support (BPMDS'07), in conjunction with CAiSE'07.

Sturm, A., Dori, D., & Shehory, O. (2006). Domain Modeling with Object-Process Methodology, Proceedings of the Eighth International Conference on Enterprise Information Systems, ICEIS (3), 144-151.

Sturm, A. & Reinhartz-Berger, I. (2004). Applying the Application-based Domain Modeling Approach to UML Structural Views, the 23rd International Conference on Conceptual Modeling (ER'2004), Lecture Notes in Computer Science 3288, 766-779.

Sunye, G., Pollet, D., Le Taraon, Y. & Jezkel J.-M. (2001). Refactoring UML models. In Proceedings of UML 2001, Lecture Notes In Computer Science 2185, 134-148.

Tersus, www.tersus.com, 2006.

Thomas, O. (2005). Understanding the Term Reference Model in Information Systems Research: History, Literature Analysis and Explanation, Proceedings of First International Workshop on Business Process Reference Models, Nancy, France, 16-28.

Tilley, S. & Huang, S. (2003). A qualitative assessment of the efficacy of UML diagrams as a form of graphical documentation in aiding program understanding, Proceedings of the 21st annual international conference on Documentation, San Francisco, CA, 184-191.

Valerio, A., Giancarlo, S. & Massimo, F. (1997). Domain analysis and framework-based software development, ACM SIGAPP Applied Computing Review, 5 (2), 4-15.

Whittle, J. (2000). Formal approaches to systems analysis using UML: An overview. Journal of Database Management, 11(4), 4-13.

Yanowitz, K. L. (2001). Transfer of structure-related and arbitrary information in analogical reasoning. The Psychological Record.

APPENDIX A: THE ADOM-UML MODEL OF THE PCS DOMAIN

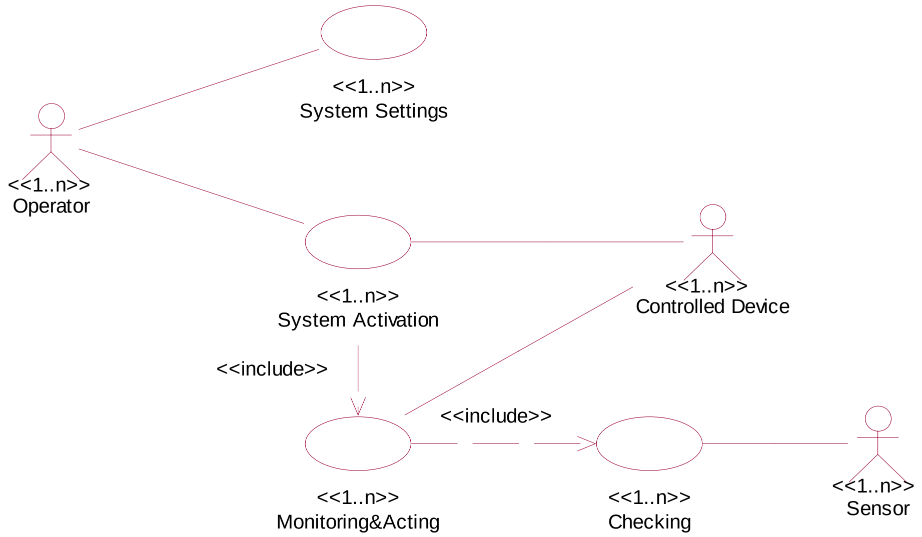

Figure 1. The PCS domain model in ADOM-UML: a use case diagram.

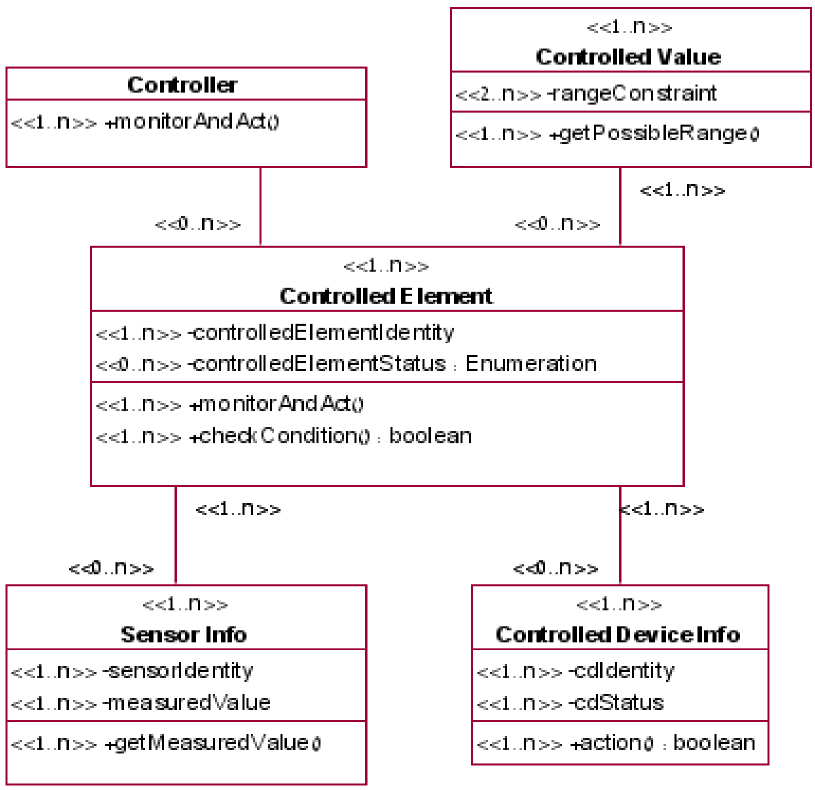

Figure 2. The PCS domain model in ADOM-UML: a class diagram.

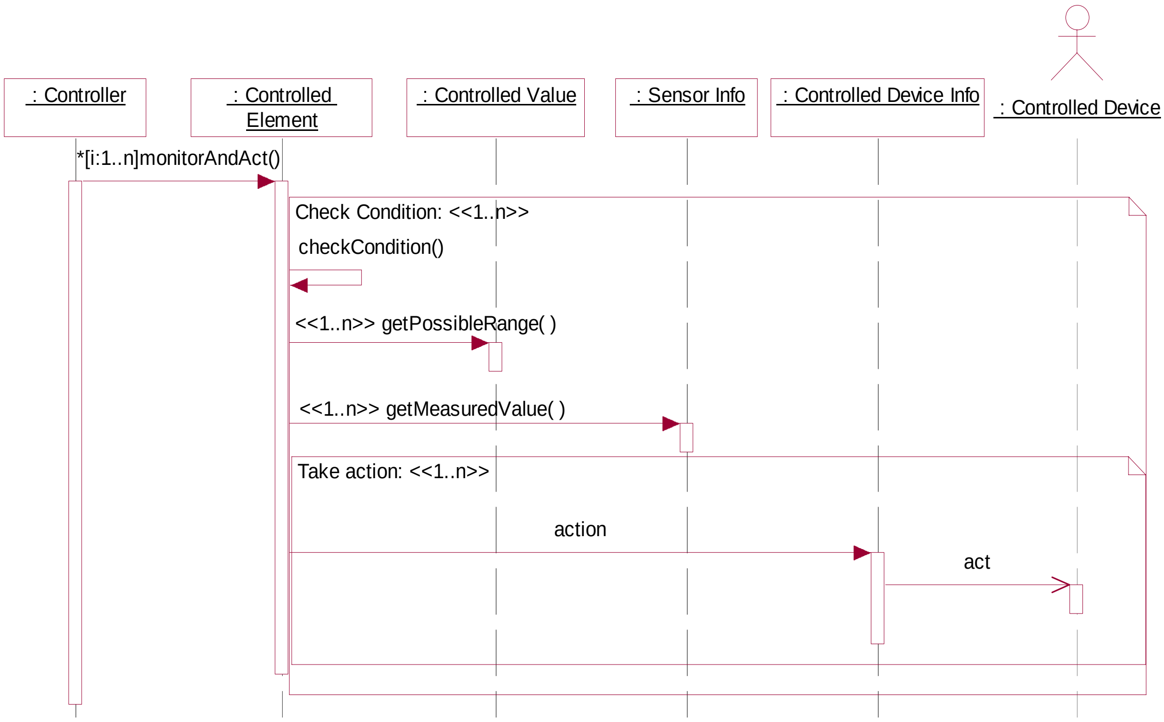

Figure 3. The PCS domain model in ADOM-UML: a sequence diagram of monitoring & acting.

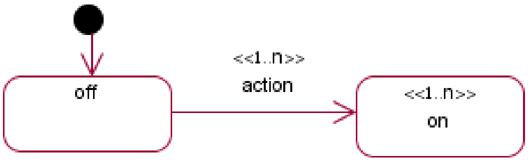

Figure 4. The PCS domain model in ADOM-UML: a statechart diagram of a controlled device.

APPENDIX B: THE ADOM-UML MODEL OF THE HCC APPLICATION

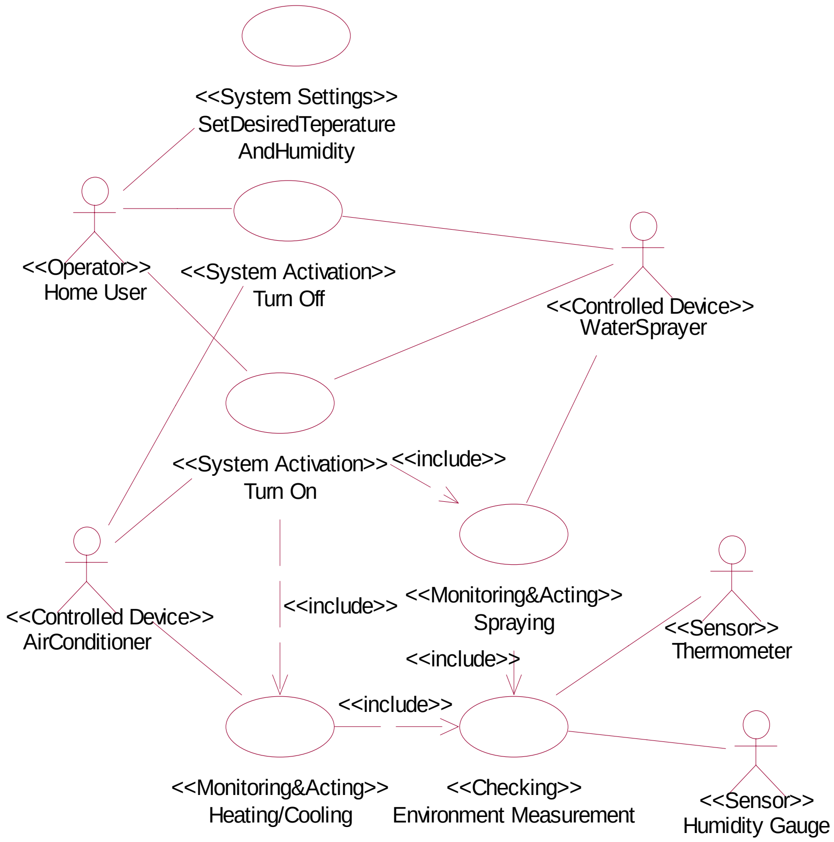

Figure 5. The HCC application model in ADOM-UML: a use case diagram.

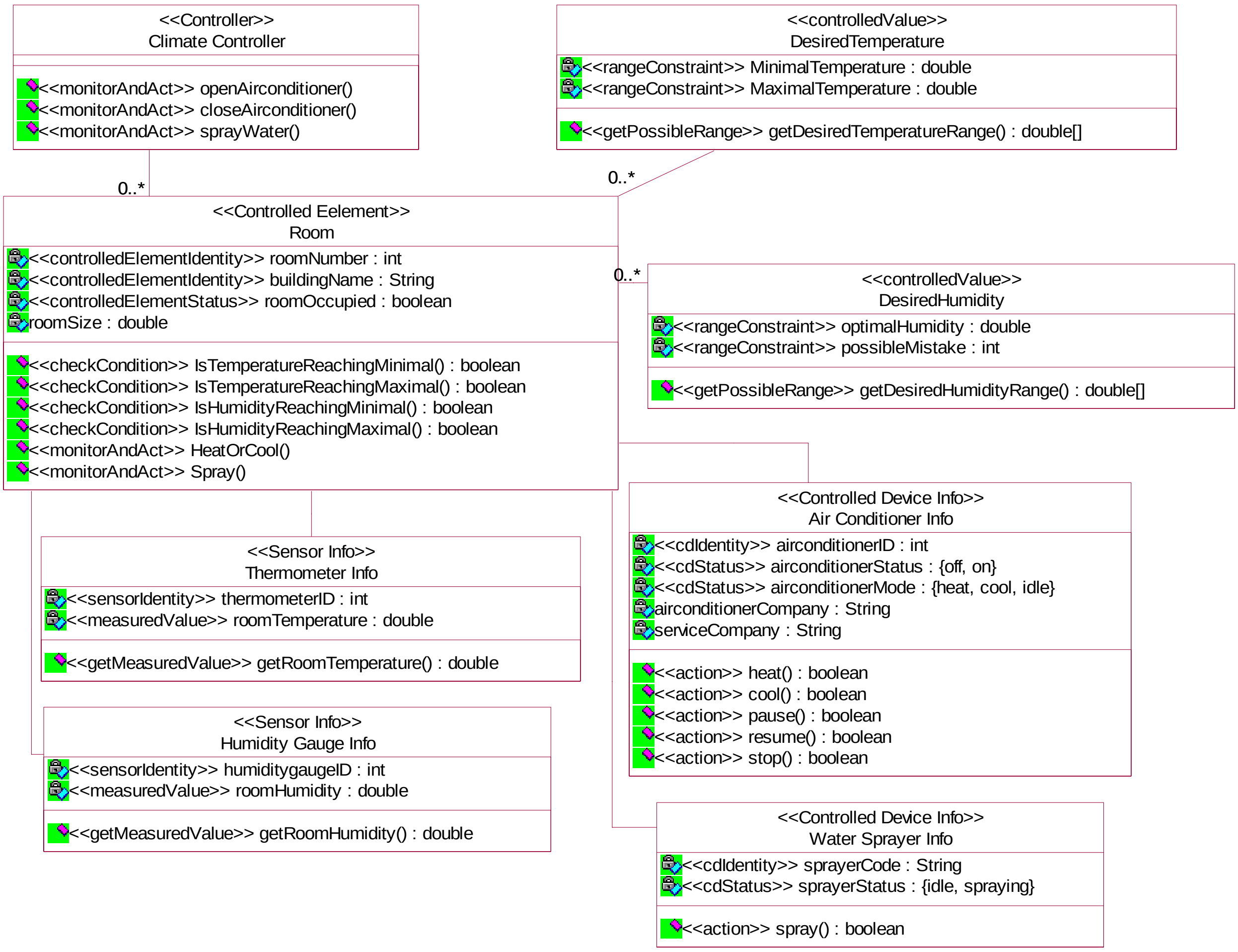

Figure 6. The HCC application model in ADOM-UML: a class diagram.

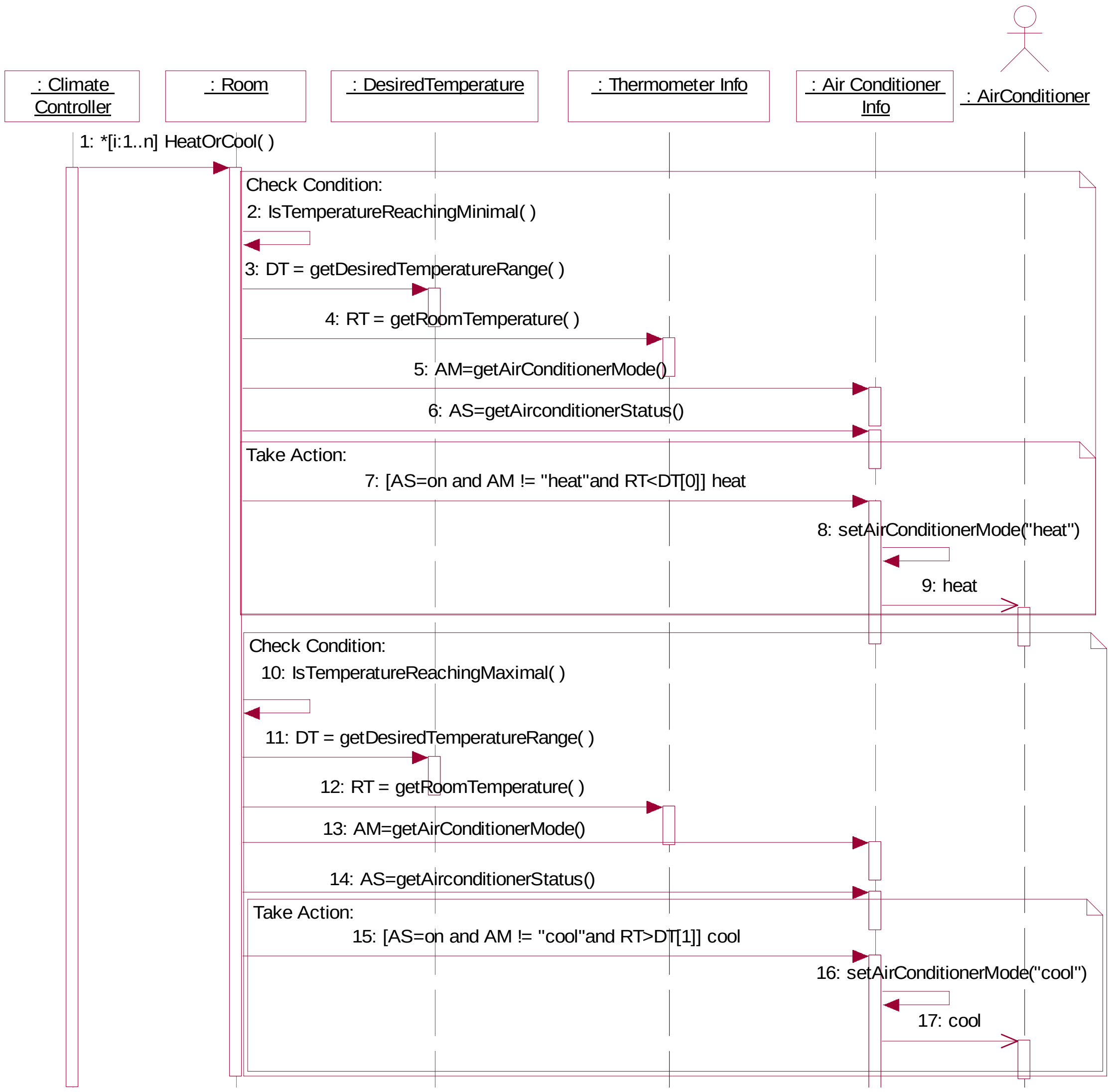

Figure 7. The HCC application model in ADOM-UML: a sequence diagram of heating/cooling.

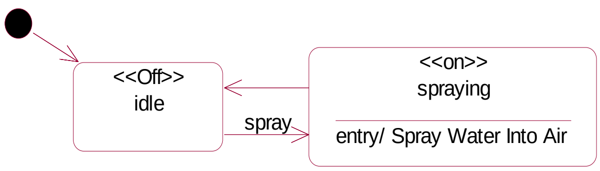

Figure 8. The HCC application model in ADOM-UML: a statechart diagram of a water sprayer.

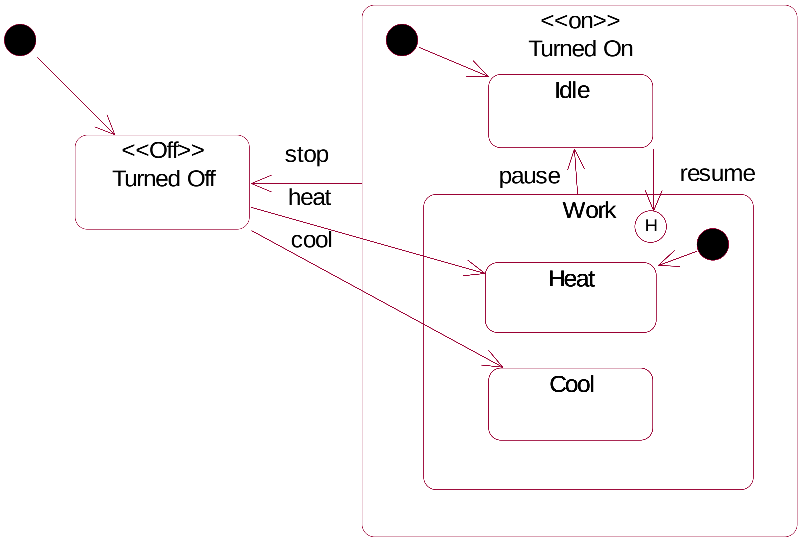

Figure 9. The HCC application model in ADOM-UML: a statechart diagram of an air-conditioner.

APPENDIX C: THE ADOM-UML MODEL OF THE WLC APPLICATION

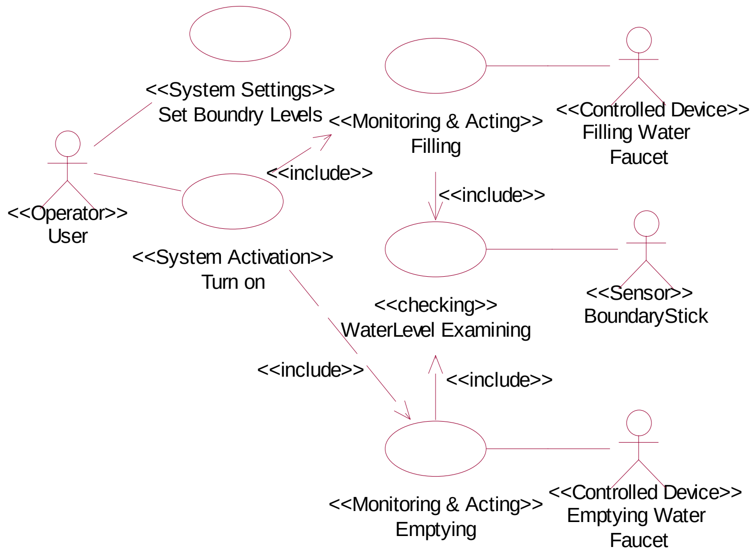

Figure 10. The WLC application model in ADOM-UML: a use case diagram.

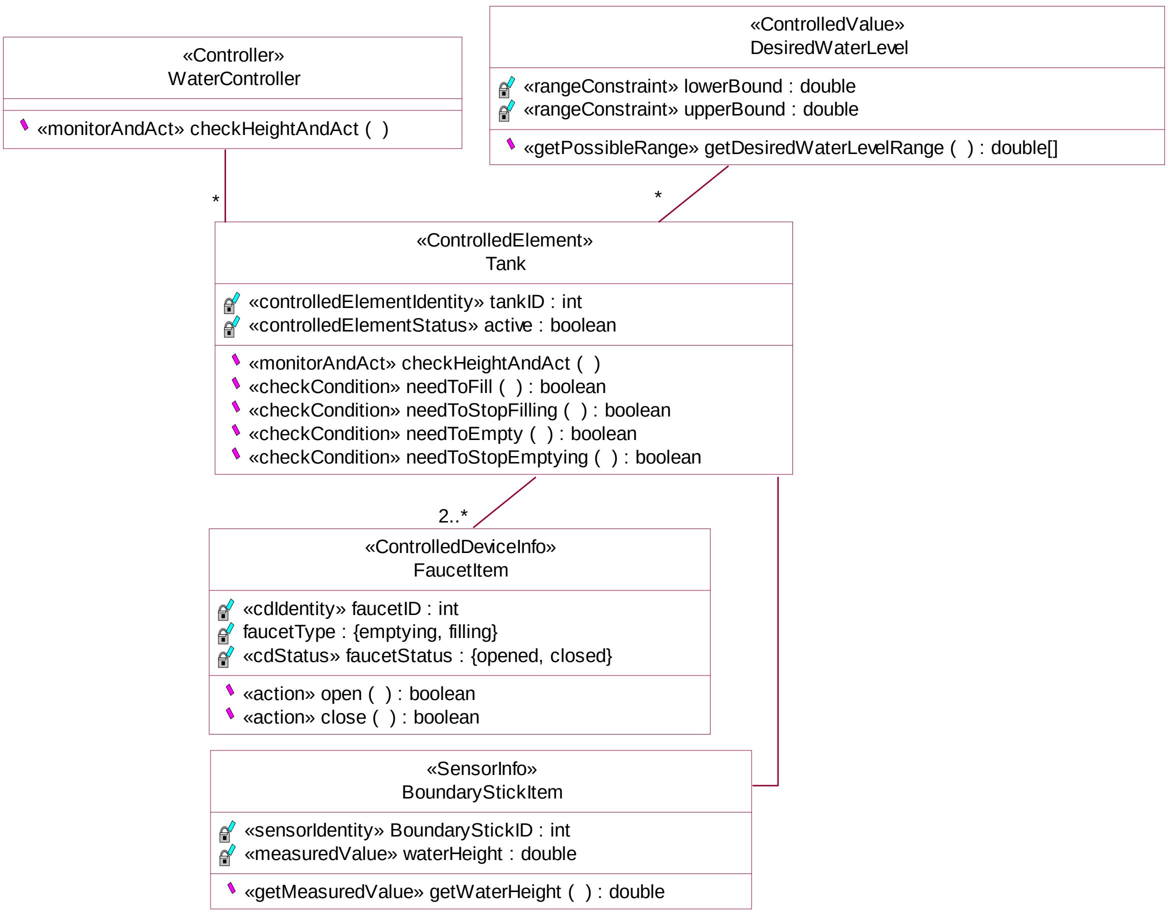

Figure 11. The WLC application model in ADOM-UML: a class diagram.

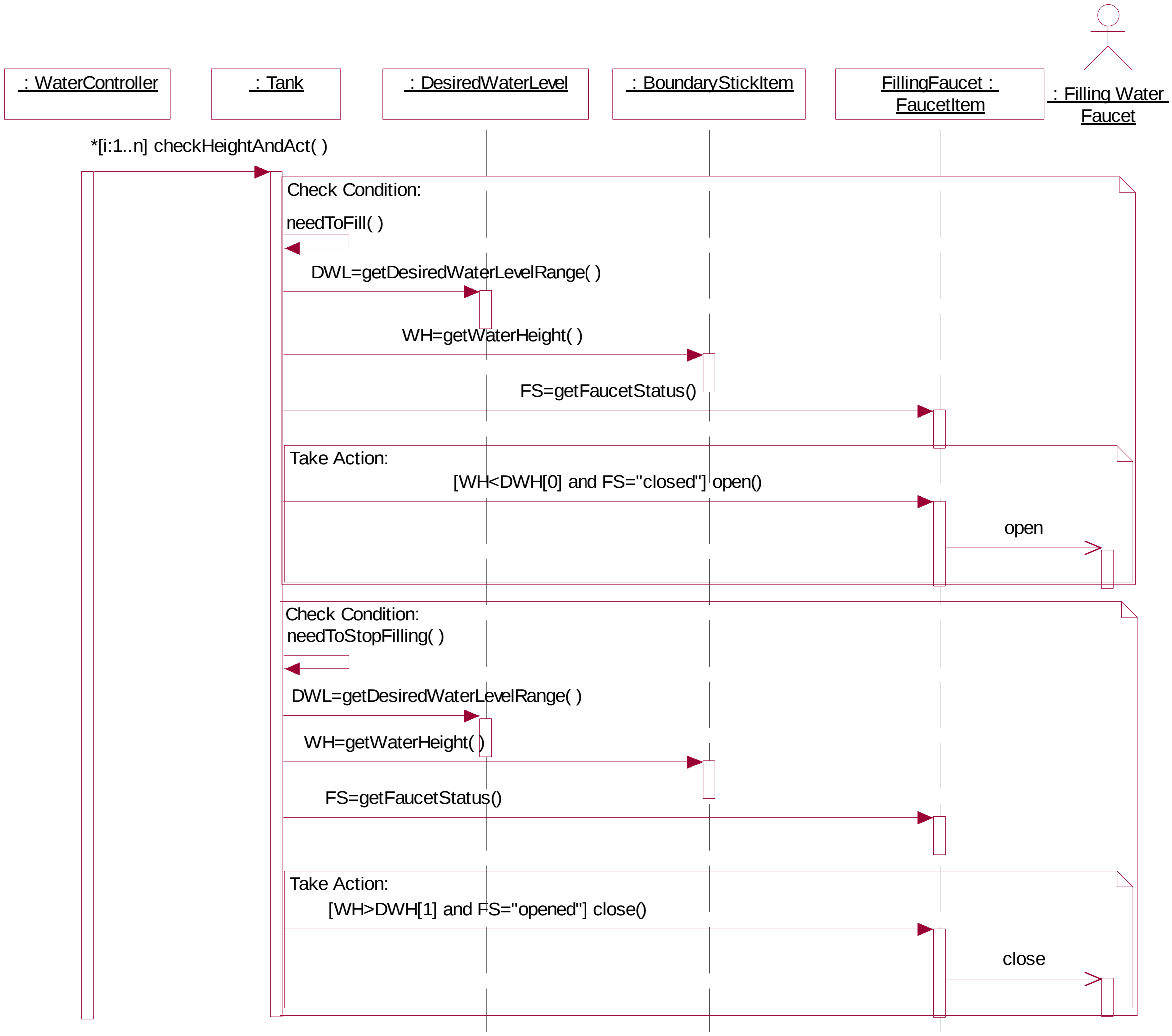

Figure 12. The WLC application model in ADOM-UML: a sequence diagram of filling/emptying.

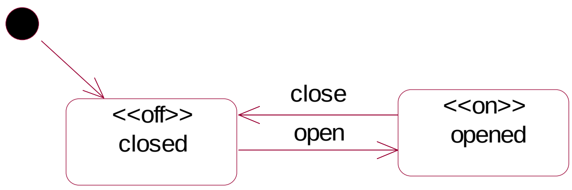

Figure 13. The WLC application model in ADOM-UML: a statechart diagram of a faucet.

APPENDIX D. THE EXPERIMENT QUESTIONS

Part 1: The HCC comprehension questions and expected answers

For each statement, state weather it is true or false and shortly explain why.

1. There are two types of devices that are controlled by the system.
   True – air-conditioners and water sprayers.
2. The system checks its sensor data, the thermometer and the humidity gauge, only through the Heating/Cooling use case.
   False – also from spraying.
3. The only possibility for the home user to activate the system is by turning it on, in addition to setting the desired temperature and humidity.
   False – also turning off.
4. According to the model, it can certainly be determined that a room is uniquely identified by its room number.
   False – there is no evidence for it in the model.
5. There are three controlled values that are controlled by the system.
   False – only temperature and humidity.
6. There can be a situation in which the water sprayer is working and the air-conditioner is not.
   True – there is no contradiction to any specification.
7. In each situation when the air-conditioner is on and the room temperature is lower than the lowest bound of the desired temperature, the heating operation of the air-conditioner is activated.
   False – only if the air-conditioner did not heat before.
8. In case that the air-conditioner is on, it cools or heats.
   False – it can be idle.
9. There are at least two sensors in each room.
   True – one thermometer and one humidity gauge.

Part 2: The WLS modeling question (abbreviated summary)

The Water Level Control (WLC) application, similarly to the HCC system, belongs to a domain of Process Control Systems (PCS). Its purpose is to monitor and control the water levels in tanks in order to ensure that the actual water level is always in the closed range [Lowest Limit, Highest Limit]. The values of the lowest and highest limits are defined per water tank and are configurable. In each tank, a boundary stick which measures the actual height of the water in the tank is installed. The tank is also coupled to filling and emptying faucets which respectively inject and drain water when the water height in the tank reaches its lowest or highest desirable limits.

You are requested to provide the following four diagrams for the WLC application:

(1) The system use case diagram
(2) The system class diagram
(3) A sequence diagram of tank filling
(4) A statechart of a water faucet

Iris Reinhartz-Berger received her B.Sc. degree in Applied Mathematics and Computer Science from the Technion, Israel Institute of Technology in 1994. She obtained a M.Sc. degree in 1999 and a PhD in 2003 in Information Management Engineering from the Technion, Israel Institute of Technology. Her M.Sc. and PhD dissertations dealt with improving various development phases, mainly design and implementation, in Object-Process Methodology (OPM). She is currently a faculty member at the Department of Management Information Systems, Haifa University, Israel. Her research interests include conceptual modeling, modeling languages and techniques for analysis and design, domain analysis, development processes, and methodologies. Her work has been published in journals and international conferences.

Arnon Sturm is a faculty member within the department of Information Systems Engineering at Ben-Gurion University. He obtained a M.Sc. degree in 1999 and a PhD in 2004 in Information Management Engineering from the Technion, Israel Institute of Technology. His dissertations dealt with applying the Object-Process Methodology (OPM) for generating database schemata, modeling multi-agent systems, and supporting domain modeling. His research concentrates on software engineering and in particular domain engineering and software development methods. Arnon has 10 years of teaching experience, including software engineering courses. Prior to his studies, Arnon has gained extensive experience in developing software systems in the industry. He also served as a member of a software engineering team that addressed problems of software development.