Domain Modeling with Object-Process Methodology

Arnon Sturm[1], Dov Dori[2], Onn Shehory[3]

[1]Department of Information Systems Engineering, Ben Gurion University of the Negev, Beer Sheva, 84105, Israel
sturm@bgu.ac.il
[2] Faculty of Industrial Engineering and Management, Technion – Israel Institute of Technology, Haifa, 32000, Israel
dori@ie.technion.ac.il
[3] IBM Haifa Research Lab, Haifa University Campus, Haifa, 31905, Israel
onn@il.ibm.com

Abstract. Domain engineering can simplify the development of software systems in specific domains. During domain analysis, the first step of domain engineering, the domain is modeled in a reusable manner. Most domain analysis approaches suffer from low accessibility, limited expressiveness, and weak formality. In this paper we present a formal, accessible and expressive approach to domain analysis. We do that by extending Object-Process Methodology (OPM) to support domain analysis. We performed an experiment to verify that the proposed extension improves the model quality compared to quality arrived at without the extension. Our experimental results show that, when presented with a set of requirements, subjects that used OPM with the domain analysis extension arrived at a system model which is better than the system model arrived at by subjects that used OPM alone.

1. Introduction

Domain engineering is concerned with building reusable software core assets and components in a specific domain of human interest [1,3]. Software reuse is viewed as a way of reducing product cycle time, thereby allowing industry to quickly deliver new products to the market [13]. Software reuse, of which domain engineering is an important factor, has therefore become a major goal for many organizations who seek to shorten time-to-market.

Domain engineering activities include domain analysis, domain design, and domain implementation. In this paper, we focus on domain analysis and how to best use it for modeling applications within a specific domain. Domain analysis can be defined as a process by which information used in developing software systems in a specific domain is identified, captured, and organized with the purpose of making it reusable when creating new systems in that domain. Domain analysis concerns the identification of a domain (or a set of related domains) and capturing the domain ontology and its variations within the domain. Subsequent stages of domain engineering, namely domain design and domain implementation are concerned with mechanisms for translating the requirements into systems that are made up of components with the intent

of reusing these components to the highest extent possible. In a more refined formulation, domain analysis is the activity of identifying objects and operations of a class of similar systems in a particular domain [29]. Domain analysis should "carefully bound the domain being considered, consider commonalities and differences of the systems in the domain, organize an understanding of the relationships between the various elements in the domain, and represent this understanding in a useful way" [1]. Domain analysis may be followed by the construction of a generic, reusable code and even a domain code generator [2].

Several methods have been developed to support domain analysis, but these methods suffer from the following weaknesses. (1) They lack formality, rendering validation of a domain-specific application against its domain model difficult to perform. (2) They require the use of several views for both domain specification and application specification, resulting in limited accessibility. (3) They address primarily the static characteristics and constraints of the domain, but their treatment of the domain's dynamic aspect is limited.

The Application-based Domain Modeling (ADOM) approach [23, 27] addresses some of the above mentioned problems. This approach treats a domain as a bona fide large application that needs to be modeled before systems in that domain are specified and designed. The domain structure and behavior thus modeled serve to define and enforce static and dynamic constraints on models of application in that domain. The ADOM approach consists of three-layers: (1) the language layer, which handles modeling language ontologies and their constraints, (2) the domain layer, which holds the building elements of domains and the relations among them, and (3) the application layer, which consists of domain-specific systems. The ADOM approach further defines dependency and enforcement relations between these layers. While ADOM builds on UML as the modeling language, its developers note that it can be applied using other modeling language as well.

In this paper, we validate the suitability of the ADOM approach to Object-Process Methodology (OPM) [5], which is an integrated approach to the study and development of systems. As a general-purpose system modeling method, OPM has been used to model systems in various domains, including pattern recognition in mechanical engineering drawings [6], computer integrated manufacturing documentation and inspection [7], and web application [20]. These systems were modeled without first devising a domain-specific ontology infrastructure. OPM was selected as the alternative modeling technique due to its supremacy over UML with respect to comprehension and construction of system models. This has been shown experimentally in [18, 21].

The contribution of this paper is threefold. First, we extend OPM with facilities and ready-to-use domain building blocks to support domain engineering principles for developing domain-specific applications. These facilities make OPM more accessible and efficient for modeling domain-specific systems and products. Second, we validate the suitability of the ADOM approach to modeling languages other than UML. Third, we experiment and provide an empirical proof of the advantage of using the ADOM-OPM-based approach over using the generic version of OPM.

The rest of this paper is organized as follows. In Section 2 we review work related to domain analysis. In Section 3 we present the ADOM approach and Section 4 intro-

duces Object-Process Methodology. Section 5 introduces the ADOM-OPM extension for domain analysis and demonstrates its use within the domain of access control system. In Section 6 we describe an experiment we performed in order to establish the suitability of ADOM-OPM for domain modeling compared with OPM and report the results. Section 7 concludes with summary and future research.

2. Related Work

Research activities on domain engineering in general and domain analysis in particular, carried out over the last decade can be divided into two major categories: (1) focusing on domain artifacts and their use during application construction and (2) using the domain model as a template and infrastructure for application construction and validation.

The first category includes works such as [13], which proposes a domain analysis process that is based on multiple views. It uses Object Modeling Technique [24] notations to produce a domain-specific framework and domain-specific components. In [29] a domain analysis and framework-based software development called Sherlock that is based on Feature-Oriented Domain Analysis (FODA) [12] is proposed. FODA activities include context definition, domain characterization, data analysis and modeling, and reusable architecture definition. A specific system model makes use mostly of the reusable architecture definitions. A domain modeling method based on multi-views [11] suggests that a specific system be derived by tailoring the domain model according to the features desired in that system. In [14], an extension to UML that supports domain engineering is proposed. This extension includes a special stereotype, which indicates that a class may be altered within a specific system. The extension is demonstrated by applying it to the UML class diagram, but none of UML's dynamic diagrams is involved in this extension.

Among works in the second category, [19] proposes building reusable repositories and architectures which consist of correlated component classes, connections, constraints, and rationales. Validation of a system model makes use of the domain model, which serves as a basis for checking that no constraint has been violated. In [15] a metamodeling technique for building a domain-specific model has been demonstrated. The approach uses UML and OCL to define the syntax, semantics, and presentation of the application models following the domain metamodel. The application analysis uses the set of models from the domain metamodel, with which one can validate the consistency and integrity of the system model [4]. Alternatives for adapting UML to a specific domain include using the metamodeling technique appear in [16, 25, 28]. Having recognized the need for domain specific models, UML has been also augmented with an extension mechanism to support such models. This mechanism includes three elements: stereotypes, tag definitions, and constraints [17].

The ADOM approach [23, 27] advocates handling the domain as a regular application as will be discussed in Section 3. For standardization reasons UML serves as the modeling language. In spite of UML's prevalence, applying the OO concept in general and UML in particular to domain modeling involves drawbacks that include increased complexity, reduced accessibility, and lack of expressive power regarding

modeling of the system's behavior and correlating it with the system's static aspect [18, 21, 22, 26]. To avoid these drawbacks, in this work we elected to use Object-Process Methodology as the basis for applying the ADOM approach.

3. The Application-based Domain Modeling approach

The Application-based Domain Modeling (ADOM) approach is based on a three layered architecture: the application layer, the domain layer, and the modeling language layer. Influenced by the Meta-Object Facility (MOF) [16], the *application layer*, which is equivalent to the MOF model layer (M1), consists of models of particular applications, including their structure (scheme) and behavior. The *language layer*, which is equivalent to the MOF metamodel layer (M2), includes metamodels of modeling languages. The modeling languages may be graphical, textual, mathematical, etc. The intermediate *domain layer*, which can be labeled M1.5, consists of specifications of various domains. The ADOM architecture also enforces constraints among the different layers. Specifically, the domain layer enforces constraints on the application layer, while the language layer enforces constraints on both the application and domain layers.

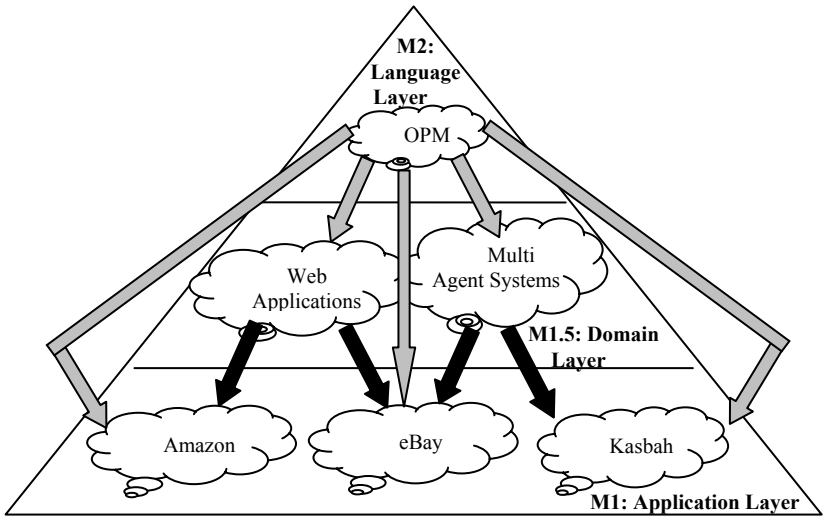

Figure 1. The Application-based Domain Modeling (ADOM) architecture

Figure 1 depicts the ADOM architecture. The application layer includes three examples of applications: *Amazon*, which is a Web-based book store, *eBay*, which is an auction site supported by agents, and *Kasbah*, which is a multi-agent electronic marketplace. Each one of these systems may have several models in different modeling languages. The domain layer in Figure 1 includes two domains: *Web applications* and *multi agent systems*, while the language layer in this example includes only one modeling language, OPM.

Figure 1 shows also the relations between the layers. The black arrows indicate constraint enforcement of the domain models on the application models, while the grey arrows indicate constraint enforcement of the language metamodels on the application and domain models.

In ADOM, domains are treated as regular applications that need to be modeled before systems in those domains can be specified and designed. Following this principle, both the domain model and the application models should use the same modeling technique to avoid incompatible notations and confused models.

4. Object-Process Methodology

Object-Process Methodology (OPM) [5] is an integrated approach to the study and development of systems in general and information systems in particular. The basic premise of the holistic OPM paradigm is that objects and processes are two types of equally important classes of things, which together describe the function, structure, and behavior of systems in a single, domain-independent model.

OPM unifies the major system lifecycle stages – initiation, development, and deployment – within one frame of reference [8], using a single diagramming tool – a set of Object-Process Diagrams (OPDs) and a corresponding subset of English, called Object-Process Language (OPL).

In OPM, system classes, class attributes, physical devices, human users, and environmental interfaces, are modeled as object classes. An object class can be either systemic (internal to the system) or environmental (external to the system). In an orthogonal manner, a class can be either physical or informatical (logical). An object class can be at one of several states, which are possible internal status values of the class objects. At any point in time, each object is at some state, and objects are transformed (generated, consumed, or affected, i.e., their state is changed) through the occurrence of a process. Unlike the object-oriented approach, behavior in OPM is not necessarily encapsulated as an operation or method within a particular object class construct. A process class can involve any number of object classes rather than be an operation of exactly one object class, as imposed by the object-oriented encapsulation principle. By allowing for the existence of stand-alone processes, rather than often having to twist reality to conform to the encapsulation principle, one can model behavior that involves several object classes intuitively and in a straightforward manner. Moreover, OPM enables the modeler to specify that some of the involved objects are transformed, while others enable the process without being transformed. The modeled behavior is integrated into the system's structure in a single model that also reflects the system dynamics in a balanced way. Processes are connected to the involved object classes through procedural links, which are classified into enabling, transformation, and event links.

OPM's built-in refinement/abstraction mechanisms, which are unfolding/folding, in-zooming/out-zooming, and state expression/suppression, help manage system complexity by controlling the visibility of details in any OPD and providing for creating new OPDs. Unfolding/folding is applied by default to objects for exposing/hiding their structural components (parts, specializations, features, or instances). In-

zooming/out-zooming is applied by default to processes for exposing/hiding their sub-process components and details of the process execution. A third scaling mechanism—state expression/suppression, enables showing or hiding the states of an object class.

4. ADOM-OPM

To implement the ADOM approach using OPM, we had to extend it with only two new features: (1) A role, which is a stereotype-like element emphasizing additional semantic for an OPM thing. Roles will be used within an application model. (2) A multiplicity indicator, which constrains the number of OPM things of some class that can be modeled in an application. The multiplicity indicator will be used within the domain model.

The rest of this section presents the domain and application layers of ADOM-OPM. This is done for the example domain of access control (AC) systems, and specifically for the Drink Vending Machine (DVM) application within the AC systems domain. Applications in the AC domain are concerned with the problem of accessing entities, objects and resources using well-defined access policies and procedures [10]. Application areas within the AC domain include all kinds of product vending machines, automated teller machines (ATM), all kinds of systems that access databases using batch and interactive interfaces, gambling machines, and local (batch, interactive) and remote access to software and hardware objects in a computer network.

The DVM application manages several machines that belong to various companies. Each machine is identified by its location and the company that owns it. The system keeps the name and telephone number of each company. Each machine works with several coin types. The products sold in each machine are identified by their name and producer. When a customer buys a drink from the system, he or she first needs to check whether the product is available and, if needed, whether coins for change are available. When the customer asks to buy a drink, the system creates a transaction, updates the relevant information and notifies the machine about the product and coins it needs to deliver. A machine operator can perform two operational activities: drinks filling and coins loading.

4.1. The ADOM-OPM Domain Layer

As noted, the domain in the ADOM approach should be modeled as a regular application. OPM is thus the modeling technique for both the domain model and the application model, and each will be constructed as an OPM model with its OPD set.

Figure 2, which depicts the system diagram (SD, top level) of the AC domain, shows that it consists of three external entities—**Client**, **Machine**, and **Maintenance Entity**, two processes—**Operate** and **Maintain**, and four system objects—**Owner**, **Company**, **Transaction**, and **Machine Info**. The symbols "m" and "+" at the right-bottom edge of some OPM things (objects and processes) indicate the multiplicity of these things within the application model. The symbol "m" indicates zero to many and

the symbol "+" indicates 1 to many. For example, at least one object of type **Client** should appear within the application model related to that domain. In addition to defining OPM things that serve as building blocks in an application in that domain, links are defined too. For example, the **Operate** process yields a **Transaction**. This constraint should hold in any applications within the AC domain.

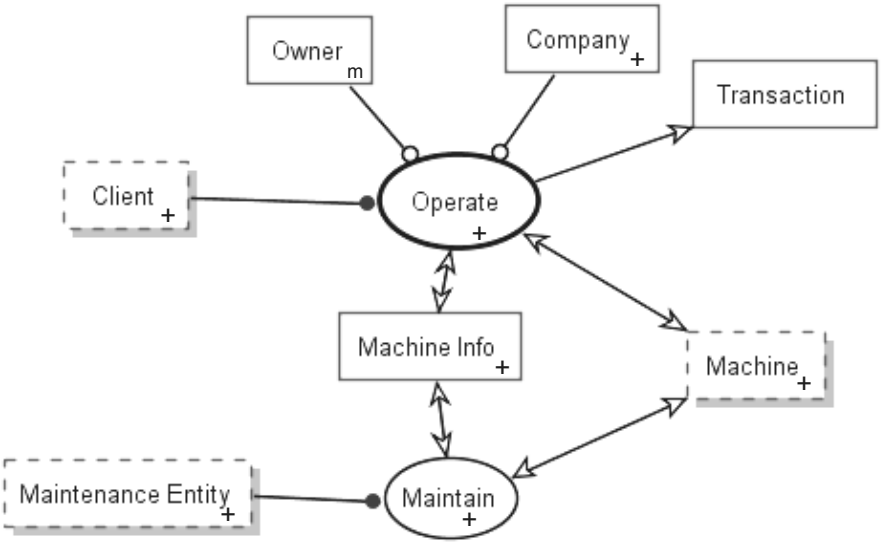

Figure 2. System Diagram of the AC domain

Machine Info is unfolded in Figure 3. **Machine Info** consists of many **Item** objects and many **Money Availability** objects. **Item** exhibits **Item Identifier** and at least one **Item Attribute** and refers to many **Transactions** and to at least one **Owner**. **Money Availability** exhibits **Money Amount** and refers to **Money Type**, which exhibits **Money Value** and at least one **Identification Sign**. **Machine Info** exhibits at least one **Machine Identifier** and an optional **Balance**. **Machine Info** refers to a **Company** and to many **Transactions**. **Company** exhibits **Company Identifier**. **Transaction** exhibits **Transaction Date** and optionally refers to **Owner** which exhibits at least one **Owner Details**.

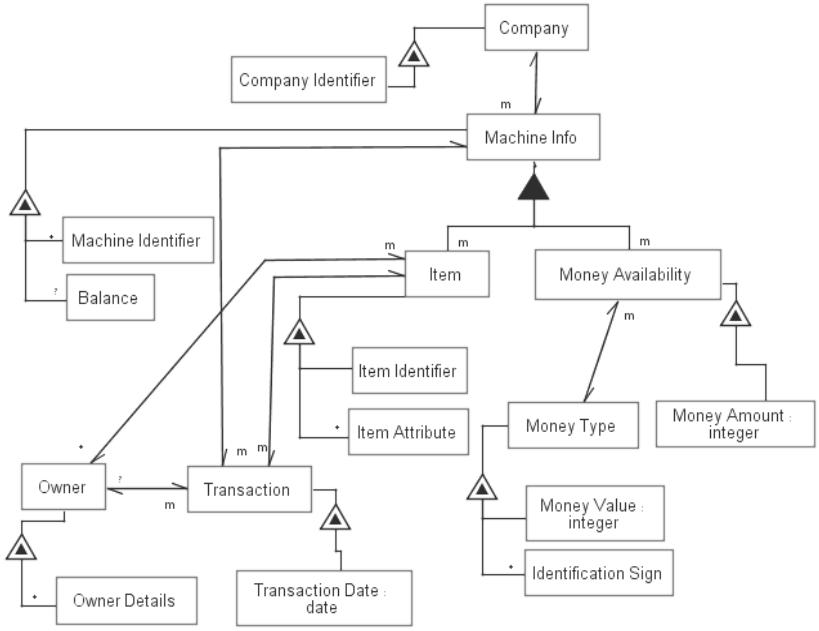

Figure 3. Machine Info Unfolded

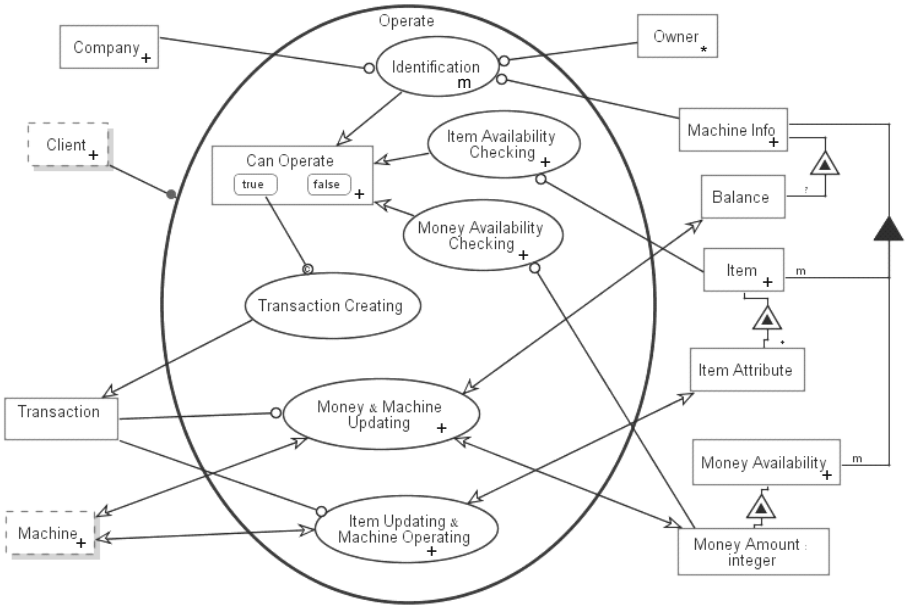

Figure 4. Operate process in-zoomed

In Figure 4 the **Operate** process is elaborated using the in-zooming scaling mechanism of OPM. The order of the processes depicted in that figure is the following:

1. An optional (as indicated by the multiplicity indicator m) **Identification** process, which requires a **Company** object, an **Owner** Object, and a **Machine Info** object and yields a **Can Operate** object;
2. At least one (as indicated by the + multiplicity indicator) **Item Availability Checking** process, which requires an **Item** object and yields a **Can Operate** object;
3. At least one **Money Availability Checking** process, which requires a **Money Amount** object and yields a **Can Operate** object;
4. A **Transaction Creating** process, which is activated if the **Can Operate** object is **true**, in which case it yields a **Transaction** object;
5. At least one **Money & Machine Updating** process, which requires the **Transaction** object and affects **Balance**, **Money Amount**, and **Machine**; and
6. At least one **Item Updating & Machine Operating** process, which requires **Transaction** and affects **Item Attribute** and **Machine**.

4.3. The ADOM-OPM Application Layer

In ADOM, the application layer uses the domain layer as a validation template. In this section we provide a specification of the Drink Vending Machine (DVM) application which is classified by [10] as belonging to the domain of access control systems. The requirements of the DVM application were presented in the beginning of Section 4.

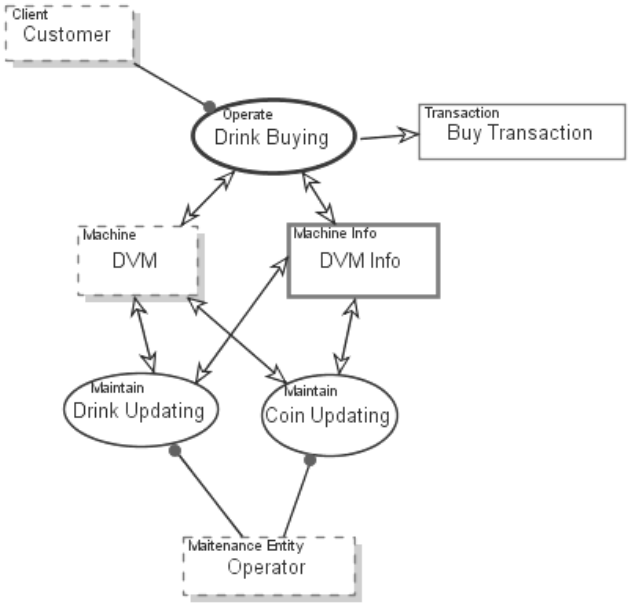

Figure 5. System diagram of the Drink Vending Machine

Figure 5 presents the system diagram of the DVM application. In the application layer model, each thing (i.e., an object or a process) is associated with a *role*. For example, the object **Customer** is associated with a *Client* role, which is an object in the domain layer model. The **Drink Buying** process is associated with the *Operate* role, a process in the domain layer model.

Note that objects that are classified as **Owner** and **Company**, which were specified in the domain layer model, do not appear in the application layer model since they are not required at the lower level OPDs of this application. This shows the ability of the ADOM-OPM approach to capture variability within a domain using the multiplicity constraints.

The system exhibits three top-level processes:

1. **Drink Buying**, which is triggered by a **Customer**, yields a **Buy Transaction**, and affects **DVM** and **DVM Info**. This process stands for the constraints that were specified with the *Operate* role in the domain layer model in Figure 2.
2. **Drink Updating**, which is triggered by the **Operator** and affects **DVM** and **DVM Info**.
3. **Coin Updating**, which is triggered by the **Operator** and affects **DVM** and **DVM Info**.

Both **Drink Updating** and **Coin Updating** conform with the constraints associated with the *Maintain* process in the domain level model.

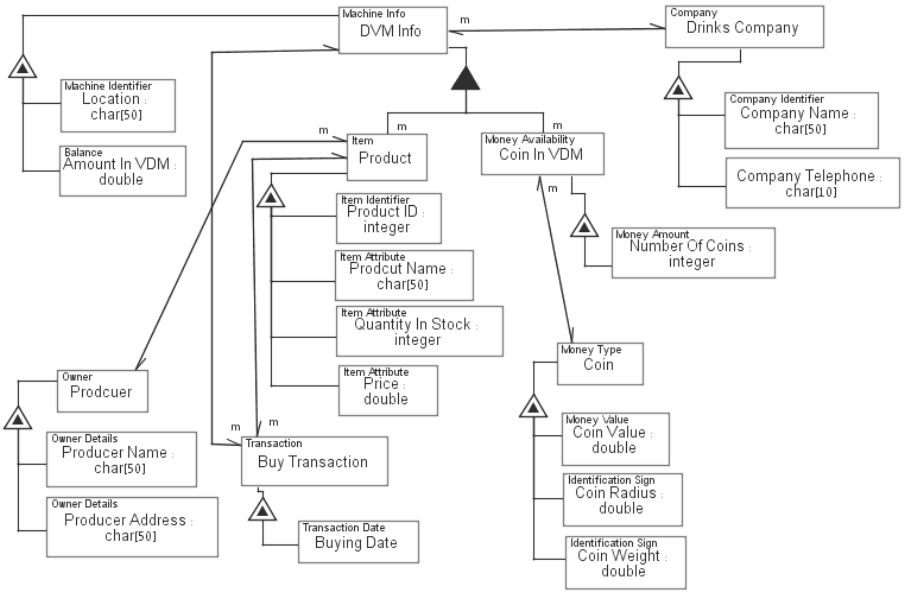

Figure 6. DVM Info Unfolded

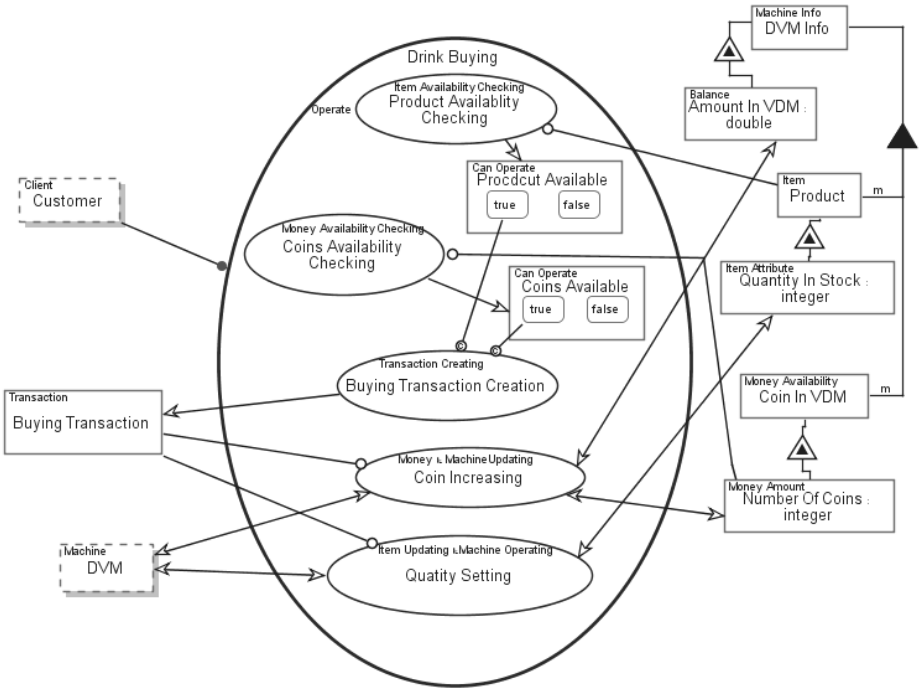

Figure 7. Drink Buying process in-zoomed

Figure 6 shows an OPD in which **DVM Info** is unfolded. This OPD relates to the OPD in Figure 3 as its validation template. The roles specified within the domain model are mapped to the application classes of both objects and processes. For example, the **Producer** labeled with the role *Owner* exhibits **Producer Name** and **Producer Address**, which are labeled with the role *Owner Details*. This relation also demonstrates how the domain layer model serves as a guideline for modeling the application.

The **Drink Buying** process, which is in-zoomed in Figure 7, follows the constraints specified in the domain layer, as described in Figure 4. Overall, the sequence of application processes follows the pattern specified in the domain layer model, yet an **Identification** process is missing, as it was specified as optional.

6. Evaluating ADOM-OPM

The ADOM-OPM approach has been applied in several domains, including multi-agent systems, discrete simulation event, resource allocation and tracking, process control, and databases. We also conducted an experiment to compare ADOM-OPM with OPM. The goal of the experiment was to determine whether modeling that is based on a domain model improves the resulting application model compared with an

application model that is developed without the support of a domain model. In this section, we present the experiment and its results.

6.1 Experiment Hypotheses

Our conjecture prior to carrying out the experiment was that an application model constructed using ADOM-OPM is more complete and more correct than the model of the same system resulting from using OPM alone. The reason for this conjecture was that the domain model in ADOM-OPM provides a framework that guides the modeler in creating the application model within the domain of discourse.

6.2 Experiment Settings

The subjects of the experiment were 120 third year students in a four-year engineering B.Sc. program at the Technion – Israel Institute of Technology, who took the course "Specification and Analysis of Information Systems" at the winter semester of the 2004-5 academic year. The students had no previous knowledge or experience in system modeling and specification. During the course, the students studied various modeling techniques, including Data Flow Diagram (DFD), UML, Statecharts, and OPM. The last lecture was devoted to the ADOM approach and its application in UML and OPM.

The experiment took place during the final examination of the course. The examination consisted of three questions relating to different domains. In each question the students were provided with application requirements similar to the requirements for the DVM application in Section 4. We had three different examination versions, such that in each question (domain) about half of the students were also provided with the OPM-based domain layer model.

The students were divided arbitrarily into three groups, labeled V1, V2, and V3, and each group responded to a different examination version. Each version included one question with a domain model and one question without a domain model. The distribution of students into the three groups and the three domains (questions) is given in Table 1, where the numbers of students who responded to each question in each version appear in parenthesis.

Table 1. Students' distribution by exam version and domain (question)

Exam Version / Domain	V1	V2	V3
Resource Allocation and Tracking (RAT)	OPM (32)		ADOM-OPM (40)
Process Control (PC)		ADOM-OPM (38)	OPM (28)
Access Control (AC)	ADOM-OPM (41)	OPM (32)	

6.3 Experiment Results

All the questions were graded by the course staff, two teaching assistants and the first author of this paper. Each one of the graders checked a question in one domain for all students according to a pre-defined set of criteria. Each question could score up to 34 points. Table 2 summarizes the average scores students achieved for each question in OPM and in ADOM-OPM.

Table 2. OPM vs. ADOM-OPM scores

Method \ Domain	RAT	PC	AC	Total
OPM	23.06	27.07	25.06	25.6
ADOM-OPM	25.00	30.64	28.19	27.55
Significance	$p<0.05$	$p<0.01$	$P<0.02$	$p<0.01$

Table 2 clearly shows that using the ADOM-OPM the students achieved better results than with OPM alone, and these results are domain independent. Performing a mean comparison statistical analysis we found that the differences between the two methods were significant. This confirms our conjecture regarding the benefits of modeling with ADOM-OPM compared with generic OPM modeling. Examining the results in detail we found out that the models done using ADOM-OPM scored better than models done with OPM alone in terms of correctness of objects, processes, and links and in terms of model completeness.

6. Summary

We have extended Object-Process Methodology (OPM) to handle application domain modeling (ADOM) approach. The OPM extension includes roles, which are stereotypes-like elements, and multiplicity indicators. We demonstrated the use of the resulting ADOM-OPM approach by applying it to the domain of access control systems and a corresponding application—the drink vending machine. Finally, we examined the ADOM-OPM approach via a controlled experiment and established that it helps create better models than those obtained using OPM alone.

The ADOM-OPM approach features the following advantages:

1. The multiple view problem: OPM supports system specification in a single, unifying view, or diagram type. Since a domain is modeled just like an application within a domain, domain modeling benefits from all the advantages of OPM, including its single view, the combination of formality with intuition, and the bimodal graphic-textual representation (not discussed in this paper).
2. Relationships between the domain and application models: The ADOM-OPM approach utilizes the domain model while modeling the application in the following ways: (1) labeling of the application model entities with roles defined in the domain model; and (2) validating the relationships among the application model

elements (entities and links) according to the thing roles and link constraints defined in the domain model.
3. The models incompatibility problem: both the domain and the application OPM models use the same notations and semantics, so no mental model transformation is needed.

Moving forward from domain analysis, domain design in OPM is similar to domain analysis, as it employs the same terminology while deepening the level of details and shifting the focus from the problem domain to the solution domain. The transformation to domain implementation can be done using the Generic Code Generator (GCG) [22] associated with OPCAT [9]. Utilizing the GCG and roles within a domain can be a basis for developing infrastructure components and using them to generate applications.

The implementation of the ADOM-OPM analysis approach is currently being integrated into OPCAT. We plan to extend the application model so that it can be based on more than one domain model. We also intend to experimentally compare the ADOM-OPM approach with ADOM-UML approach.

Acknowledgement

The authors wish to thank Dr. Iris Reinhartz-Berger for her insightful remarks and comments and for her help in designing the experiment.

References

1. Carnegie Mellon - Software Engineering Institute, “Domain Engineering: A Model-Based Approach”, http://www.sei.cmu.edu/domain-engineering/, 2002.
2. de Champeaux D., Lea D., and Faure P., Object-Oriented System Development, Addison Wesley, 1993.
3. Cleaveland C., “Domain Engineering”, http://craigc.com/cs/de.html, 2002.
4. Davis J., “Model Integrated Computing: A Framework for Creating Domain Specific Design Environments”, The Sixth World Multiconference on Systems, Cybernetics, and Informatics (SCI), 2002.
5. Dori D., Object-Process Methodology - A Holistic Systems Paradigm, Springer Verlag, 2002.
6. Dori D., “Representing Pattern Recognition-Embedded Systems through Object-Process Diagrams: the Case of Machine Drawing Understanding System”, Pattern Recognition Letters, 16(4), pp. 374-384, 1995.
7. Dori D., “Object-Process Analysis of Computer Integrated Manufacturing Documentation and Inspection”, International Journal of Computer Integrated Manufacturing, 9(5), pp. 339-353, 1996.
8. Dori D. and Reinhartz-Berger I., “An OPM-Based Metamodel of System Development Process”, Proceedings of Twenty Second International Conference on Conceptual Modeling (ER), 2003.

9. Dori D., Reinhartz-Beger I., and Sturm A., “OPCAT – A Bimodal CASE Tool for Object-Process Based System Development”, The fifth International Conference On Enterprise Information Systems (ICEIS), 2003.
10. Duffy, D. J., “Domain Architectures: Models and Architectures for UML Applications”, John Wiley & Sons, 2004.
11. Gomaa E. and Kerschberg L., “Domain Modeling for Software Reuse and Evolution”, Proceedings of Computer Assisted Software Engineering Workshop (CASE 95), 1995.
12. Kang K., Cohen S., Hess J., Novak W., and Peterson A., “Feature-Oriented Domain Analysis (FODA) Feasibility Study”, CMU/SEI-90-TR-021 ADA235785, 1990.
13. Meekel J., Horton T. B., France R. B., Mellone C., and Dalvi S., “From domain models to architecture frameworks”, Proceedings of the 1997 symposium on Software reusability, p.75-80, 1997.
14. Morisio M., Travassos G. H., and Stark M., “Extending UML to Support Domain Analysis”, Proceedings of the Fifth IEEE International Conference on Automated Software Engineering, pp. 321-324, 2000.
15. Nordstrom G., Sztipanovits J., Karsai G., and Ledeczi A., “Metamodeling - Rapid Design and Evolution of Domain-Specific Modeling Environments”, Proceedings of the IEEE Sixth Symposium on Engineering Computer-Based Systems (ECBS), pp. 68-74, 1999.
16. OMG -MOF, “Meta-Object Facility (MOF™)”, version 1.4, 2002.
17. OMG-UML, “The Unified Modeling Language (UML™)”, version 1.5, 2003.
18. Peleg M. and Dori D., “The Model Multiplicity Problem: Experimenting with Real-Time Specification Methods”, IEEE Transaction on Software Engineering, 26 (8), pp. 742-759, 2000.
19. Petro J. J., Peterson A. S., and Ruby W. F., “In-Transit Visibility Modernization Domain Modeling Report Comprehensive Approach to Reusable Defense Software” (STARS-VC-H002a/001/00). Fairmont, WV: Comprehensive Approach to Reusable Defense Software, 1995.
20. Reinhartz-Berger I., Katz S., and Dori D., “OPM/Web - Object-Process Methodology for Developing Web Applications”, Annals on Software Engineering - Special Issue on OO Web-based Software Engineering, pp. 141-161, 2002.
21. Reinhartz-Berger I. and Dori D., “OPM vs. UML – Experimenting Comprehension and Construction of Web Application Models”, Empirical Software Engineering Journal, 10 (1), pp. 57-80, 2005.
22. Reinhartz-Berger I. and Dori D., “Object-Process Methodology (OPM) vs. UML: A Code Generation Perspective”, EMMSAD, 2004.
23. Reinhartz-Berger I. and Sturm A., “Behavioral Domain Analysis – The Application-based Domain Modeling Approach”, the 7th International Conference on the Unified Modeling Language (UML'2004), LNCS 3273, pp. 410-424, 2004.
24. Rumbaugh J., Blaha M., Premerlani W., Eddy F., and Lorensen W., “Object-Oriented Modeling and Design”, Prentice-Hall International Inc., 1991
25. Schleicher A. and Westfechtel B., “Beyond Stereotyping: Metamodeling Approaches for the UML”, Proceedings of the Thirty Fourth Annual Hawaii International Conference on System Sciences, pp. 1243-1252, 2001.
26. Siau K. and Cao Q., “Unified Modeling Language: A Complexity Analysis”, Journal of Database Management, 12 (1), pp. 26-34, 2001.
27. Sturm A. and Reinhartz-Berger I., “Applying the Application-based Domain Modeling Approach to UML Structural Views”, the 23rd International Conference on Conceptual Modeling (ER'2004), LNCS 3288, pp. 766-779, 2004.

28. Terrasse M. and Savonnet M., “Formalization of the UML Metamodel: An Approach Based Upon the Four-Layer Metamodeling Architecture”, Proceedings of the Fourteenth ECOOP Workshop on Defining a Precise Semantics for UML, 2000.
29. Valerio A., Succi Giancarlo, and Fenaroli Massimo, “Domain analysis and framework-based software development”, ACM SIGAPP Applied Computing Review, 5 (2), 1997.