Guiding System Modelers in Multi View Environments: A Domain Engineering Approach

Arnon Sturm

Department of Information Systems Engineering

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

sturm@bgu.ac.il

Abstract. Nowadays, due to system complexity, it is well understood that system analysis and design should be done at various levels of abstraction via various perspectives. This situation of multiple views over a system causes inconsistencies within the system model, which reduces the model maturity for the next stages (e.g., implementation and testing). In this paper, we address this gap by integrating two approaches from the area of domain analysis: feature modeling and the Application-based DOmain Modeling Approach (ADOM). Using the integrated approach we provide the system developers with guidelines of how to construct a multi-view system model in a way that the various views will be synchronized and adhered with the desired specification. In this paper we adopt UML as the modeling language and demonstrate the usage of the proposed method on multi view UML based model.

1. Introduction

As the complexity of system is increased, it is desirable to decompose the system into hierarchical parts reducing the complexity of each one of these. The analogy of that situation to modeling is the division of models into packages of different parts and of different perspectives. However, when referring to that kind of models, the model multiplicity problem is raised as appears in [11]. The authors in that paper advocate that the specification quality is reduced due to that model multiplicity problem in some of the cases. In [15] the authors had claimed the same and propose a methodology to reduce that problem. However, these works somewhat neglect the need for capturing the system at different levels of abstraction and at various viewpoints. The Unified Modeling Language (UML) has addressed this need, however, it falls short in synchronizing the various diagrams. Thus causes inconsistency among the diagrams. Inconsistency problems include syntactic and semantic errors. While syntactic rules deal with diagrams being well-formed and can be checked by CASE tools, semantic consistency which deals with the meaning of different views and the compatibility between these is difficult to check. In addition, having multiple views may become an obstacle when deciding which views a designer should choose when specify a specific application. For example, in [4] the authors survey the usage of UML and found that it is only partially used with respect to the multiple aspects specification.

In this paper we address both the designer guidance problem as well as the support for consistency in multiple-view environments problem. In particular, we utilize two approaches originated from the domain analysis era, namely, feature modeling [9], and the Application-based Domain Modeling (ADOM) [12, 16] for the task at hand.

Feature modeling enables defining the prominent and distinctive user visible characteristics of a system [8]. When referring to a domain, feature modeling captures the commonality and variability among the applications of that domain.

ADOM treats a domain as an application in its own right that needs to be modeled before systems in that domain are specified and designed, yet the entire domain is modeled as a regular application that serves as a reference to applications in that domain. The same paradigm, along with its semantics—the set of concepts, and its syntax—the set of symbols, is used for specifying domains and the applications within them. The modeled domain structure and behavior serve to define and enforce static and dynamic constraints on application models in that domain. ADOM consists of three layers: (1) the language layer, which is concerned with the underlying modeling language, pertinent ontologies, and their constraints; (2) the domain layer, which uses the language defined within the language layer to model the various domains, including the building blocks of each domain and the relationships among them; and (3) the application layer, which consists of domain-specific system models. The ADOM approach explicitly enforces constraints among the different layers: the domain layer enforces constraints on the application layer, while the language layer enforces constraints on both the application and domain layers.

We advocate that the integration of both approaches: feature modeling and ADOM leads to an increase within a systems specification quality due to the guidelines provided to developers and due to the constraints provided (and checked) within the domain model.

The rest of the paper is organized as follows. Section 2 reviews studies related to the problem of consistency in multi-view approaches and to the problem of missing guidelines for developing system specifications. Section 3 introduces the principles on which the integrated approach works and demonstrates its use. In Section 4 we conclude and set our plans for the continuous research.

2. Literature Review

Modeling complex system includes structural and behavioral aspects. For example, in UML different diagrams, or views, deal with different aspects of the system. Using different views assists in focusing on the specific developed aspect and in keeping each diagram size reasonable. However, having different views of the same system raises consistency problems between the different views. In addition, the problem of which views are required for a specific application remains unsolved.

In [14] the author provides a comprehensive survey related to consistency and integration problems occur in UML models. These originated due to the multiple view definition. Furthermore, the proposed solutions only partially addressed the aforementioned problems. In addition, that paper proposes TLOOF, which is a framework that addresses that problem. However, although providing tools for keeping consistency and avoiding integration problems, TLOOF does not provide guidelines for the system developers of which views (i.e., diagrams) are required for a specific application.

In [1] the author aims at providing guidelines for creating an analysis model that is both complete and correct. However, these guidelines refer mainly to the development process perspective neglecting the process artifacts, i.e., the models. Thus, there is no reference for handling the consistency problem.

Other approaches originated for the domain engineering area [6]. For example, the feature modeling technique has been used for that purpose. In [2] the authors propose a template based approach for mapping feature models to other models to represent variability. A similar approach appears in [10]. In [7] the proposed approach also associates features with models. The latter approaches narrow the system specification to be configurative and restrict the addition of new features to the system specification.

In [13] the authors experiment the correctness and completeness of a system UML–based specification when providing a domain model. The results of that experiment showed that when providing a domain model the completeness of the application model is increased. However, the experiment checked the various views (i.e., diagram) separately, and the subjects were asked to provide specific views. Thus, the consistency among views and integration problems were out of the scope of that work.

3. The Proposed Approach

Following the gaps aforementioned, in this paper we aim at providing an approach to enable the provisioning of guidelines for specifying applications in multi-view environments. By guidelines, we refer to what are the views that should be provided when specifying a specific application and what are the elements within these views that should be instantiated. We support two types of instantiation: specialization and configuration. In addition, we would like the approach to be able to enforce various constraints that will maintain the integrity among the various views.

In order to facilitate the proposed approach we utilize two complementary techniques: feature modeling and the ADOM approach, and refer to a model at two levels: the domain level and the application level.

3.1. The Domain Level

The domain level models provide specification guidelines, serve as a validation template, and enforce multi-view consistency. For that purpose, the domain level consists of the following:

A feature model: The feature model aims at capturing the features applicable to a specific domain. In particular, we adopt the cardinality-based version as appear in [3].

However, we used the feature models at different level of abstraction. That is, an instantiation of the feature model might be a configuration, a specialization as appear in [8] but may also include sub typing of the various features and new features can be added as well. The feature model serves as an external point of view of the system and should help the designer to capture the system internal structure (and functionality). Figure 1 presents a feature model of a Resources Allocation and Tracking (RAT) domain and its instantiation related to an elevator control system. The RAT domain consists of applications that process customer requests, allocate resources to realize those requests and monitor the status of requests at all times [5]. The elevator control system is responsible for the registration of passenger requests, allocating an elevator for that request, and handling the request.

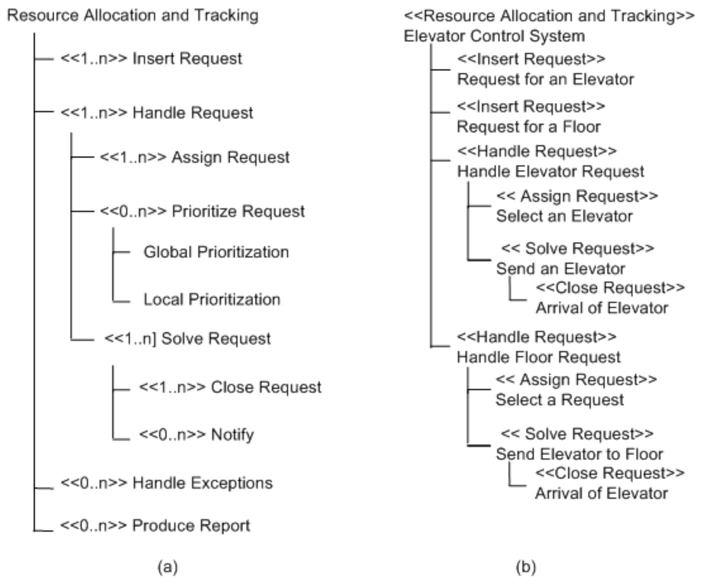

Fig. 1. A domain feature model and its instantiation within the RAT domain

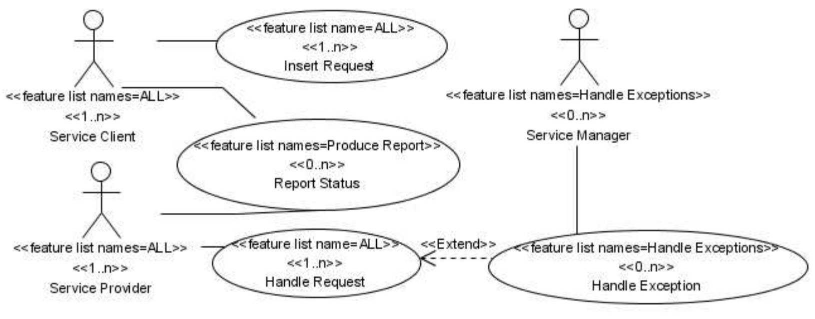
<<feature list name=ALL>>
<<1..n>>
Insert Request
<<feature list names=ALL>>
<<1..n>>
Service Client
<<feature list names=Produce Report>>
<<0..n>>
Report Status
<<feature list names=Handle Exceptions>>
<<0..n>>
Service Manager
<<feature list names=ALL>>
<<1..n>>
Service Provider
<<feature list name=ALL>>
<<1..n>>
Handle Request
<<Extend>>
<<feature list names=Handle Exceptions>>
<<0..n>>
Handle Exception

Fig. 2. A use case diagram of the RAT domain

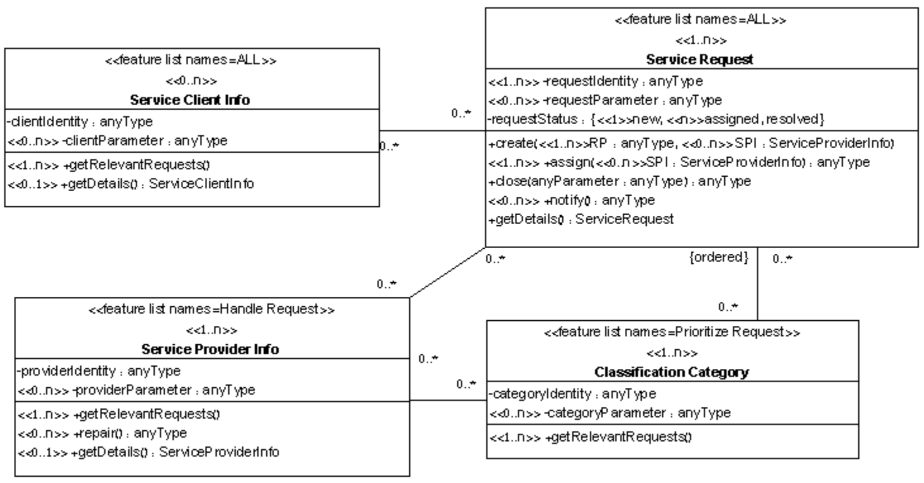
<<feature list names=ALL>>
<<0..n>>
Service Client Info
-clientIdentity : anyType
<<0..n>> -clientParameter : anyType
<<1..n>> +getRelevantRequests()
<<0..1>> +getDetails() : ServiceClientInfo
<<feature list names=ALL>>
<<1..n>>
Service Request
<<1..n>> -requestIdentity : anyType
<<0..n>> -requestParameter : anyType
-requestStatus : {<<1>>new, <<n>>assigned, resolved}
+create(<<1..n>>RP : anyType, <<0..n>>SPI : ServiceProviderInfo)
<<1..n>> +assign(<<0..n>>SPI : ServiceProviderInfo) : anyType
+close(anyParameter : anyType) : anyType
<<0..n>> +notify() : anyType
+getDetails() : ServiceRequest
0..*
0..*
0..*
{ordered}
0..*
0..*
0..*
<<feature list names=Handle Request>>
<<1..n>>
Service Provider Info
-providerIdentity : anyType
<<0..n>> -providerParameter : anyType
<<1..n>> +getRelevantRequests()
<<0..n>> +repair() : anyType
<<0..1>> +getDetails() : ServiceProviderInfo
0..*
0..*
<<feature list names=Prioritize Request>>
<<1..n>>
Classification Category
-categoryIdentity : anyType
<<0..n>> -categoryParameter : anyType
<<1..n>> +getRelevantRequests()

Fig. 3. A class diagram of the RAT domain

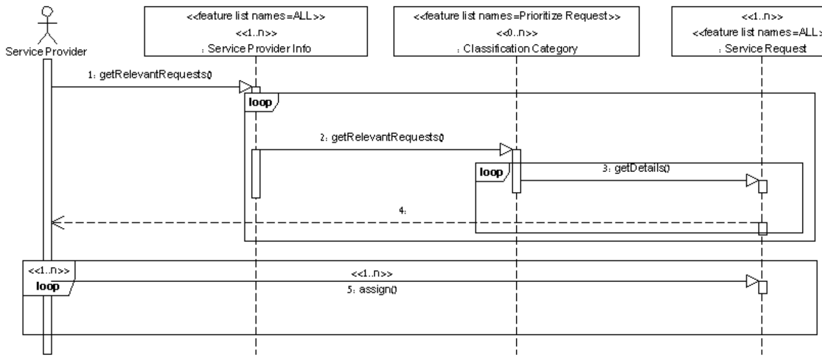

Fig. 4. A Handle a Service Request by a Service Provider SD belongs to the RAT domain

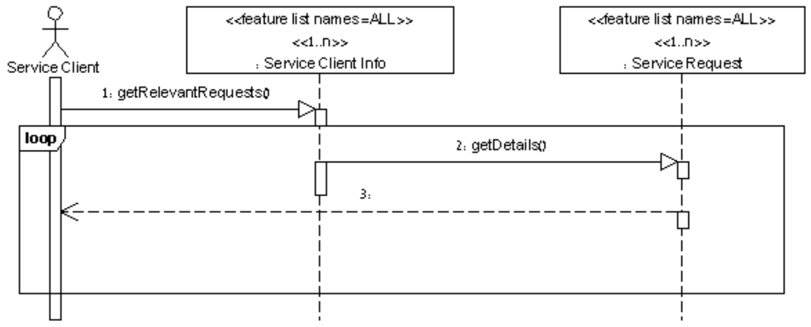

Fig. 5. A Check Service Request Status by a SD belongs to the RAT domain

***A functional (multi-views) model*:** The functional model may include structural diagrams, behavioral diagrams, dynamic behavioral diagrams, etc. In this paper, we adopt UML as the modeling language, thus the functional model may include class diagrams, use case models, sequence diagrams, etc. The functional model is represented by following the guidelines of the ADOM approach. In particular, each element within the model is associated with a multiplicity indicator which determines the number of instantiations that can be part of application models in the specific context. This is denoted by the <<min, max>> annotation. Figures 2-5 present four views of the RAT domain: a use case diagram, a class diagram, and two sequence diagrams. For example, the *Service Client* actor within the use case diagram should appear at least once in any application within the RAT domain. Similarly, the *Service Client Info* class within the class diagram is optional, yet may appear several times in applications within that domain. In addition to the multiplicity indicator suggested by the ADOM approach, the model elements should be assigned to features as well. This determines whether a model element requires attention in case a specific feature is instantiated. For example, the *Service Request* class is relevant for all features, whereas the *Classification Category* class is relevant only when having a feature of type *Prioritize Request*.
***A mapping model*:** The mapping model maps the views to features. Figure 6 specifies the mapping in the case of the RAT domain. For example, the *Main Use Case Model* should appear for all features whereas the *Check Service Request Status by a Client* sequence diagram is required only in a case of which the *Produce Report* feature is selected.

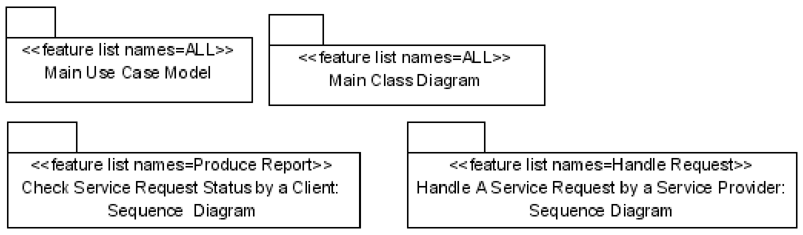
<<feature list names=ALL>>
Main Use Case Model
<<feature list names=ALL>>
Main Class Diagram
<<feature list names=Produce Report>>
Check Service Request Status by a Client: Sequence Diagram
<<feature list names=Handle Request>>
Handle A Service Request by a Service Provider: Sequence Diagram

Fig. 6. Mapping of domain views and features

3.2. The Application Level

The application level consists of models which are instantiations of the one within the domain level. For example, the feature model of the elevator control system appears in Figure 1 (b) instantiates the RAT feature model appears in Figure 1 (a). The instantiations of the elevator control system functional model appear in Figures 7-10.

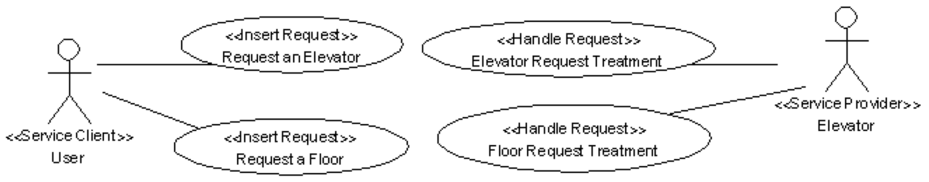

Fig. 7. A use case diagram of the elevator control system

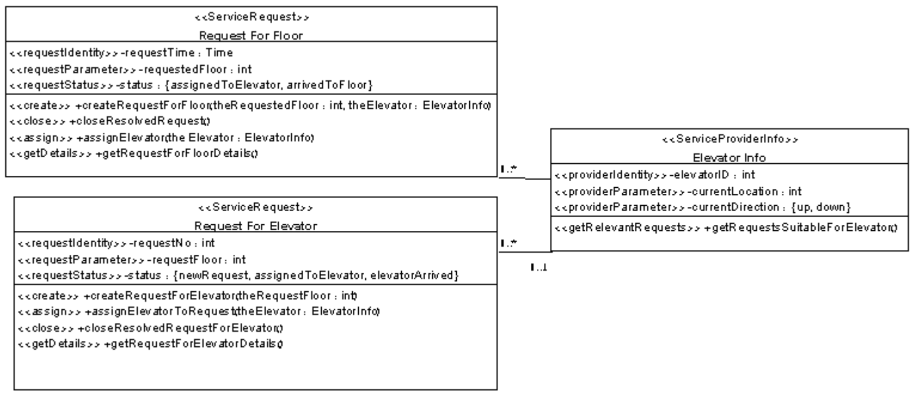
<<ServiceRequest>>
Request For Floor
<<requestIdentity>> -requestTime : Time
<<requestParameter>> -requestedFloor : int
<<requestStatus>> -status : {assignedToElevator, arrivedToFloor}
<<create>> +createRequestForFloor(theRequestedFloor : int, theElevator : ElevatorInfo)
<<close>> +closeResolvedRequest()
<<assign>> +assignElevator(the Elevator : ElevatorInfo)
<<getDetails>> +getRequestForFloorDetails()
1..*
<<ServiceProviderInfo>>
Elevator Info
<<providerIdentity>> -elevatorID : int
<<providerParameter>> -currentLocation : int
<<providerParameter>> -currentDirection : {up, down}
<<getRelevantRequests>> +getRequestsSuitableForElevator()
<<ServiceRequest>>
Request For Elevator
<<requestIdentity>> -requestNo : int
<<requestParameter>> -requestFloor : int
<<requestStatus>> -status : {newRequest, assignedToElevator, elevatorArrived}
<<create>> +createRequestForElevator(theRequestFloor : int)
<<assign>> +assignElevatorToRequest(theElevator : ElevatorInfo)
<<close>> +closeResolvedRequestForElevator()
<<getDetails>> +getRequestForElevatorDetails()
1..*
1..1

Fig. 8. A class diagram of the elevator control system

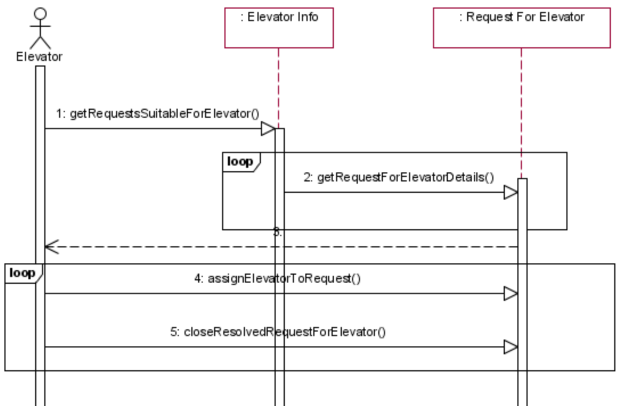
Elevator
: Elevator Info
: Request For Elevator
1: getRequestsSuitableForElevator()
loop
2: getRequestForElevatorDetails()
3:
loop
4: assignElevatorToRequest()
5: closeResolvedRequestForElevator()

Fig. 9. A Handle Request For Elevator SD

In this case each of the model elements is classified according to the domain model element. For example, the *User* actor within the use case diagram appears in Figure 7 is classified as a *Service Client*, the *Request an Elevator* use case is classified as *Insert Request*, *Request For Floor* and *Request For Elevator* classes within the class diagram appears in Figure 8 are classified as *Service Request*. Note that additional features and additional model constructs can be defined within the application model even though these were not defined in the domain model. However, these additions should not contradict constrains defined within the domain model. This capability increases the support in systems variability within the domain.

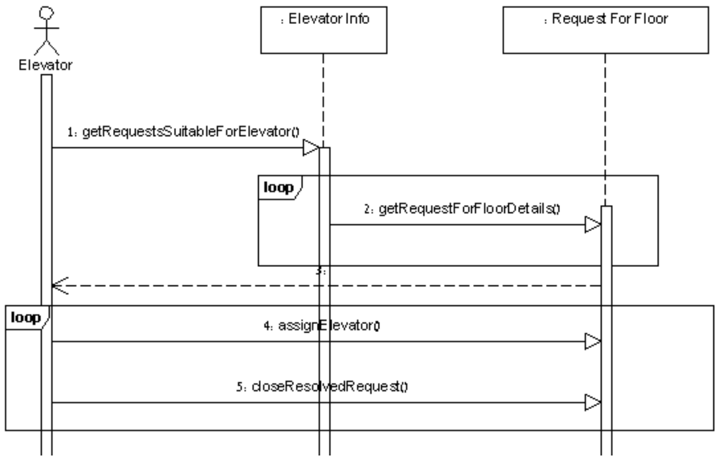
Elevator
: Elevator Info
: Request For Floor
1: getRequestsSuitableForElevator()
loop
2: getRequestForFloorDetails()
loop
4: assignElevator()
5: closeResolvedRequest()

Fig. 10. A Handle Request For Floor sequence diagram

3.3. The Modeling Paradigm

When using the proposed method we opt for the following procedure. First, the designer should determine the domain of which the system-to-be is related to. Then the domain model is retrieved and serves as a guideline as well as a validation template. Note that we expect that the domain model is well-established. Next, the designer should examine the external point of view of the system by selecting the relevant features for the system and instantiates those. This model guides the designer of which features are mandatory and which are optional. Upon selecting and instantiating the selected features (and maybe add application specific ones), the designer should follows the mapping model for the selected features in order to provide the right set of views as determined by the domain model. In the case of the elevator control system, since the selected features were *Insert Request* and *Handle Request*, the required views are the *Main Use Case Model*, the *Main Class Diagram*, and the *Handle a Service by a Service Provider* sequence diagram. Note that since there are two instantiations of the *Handle Request*, there should be two instantiations of the *Handle a Service by a Service Provider* sequence diagram as appear in Figures 9-10.

When developing each of the views, the designer should follow the mapping to features as well. For example in the class diagram of the elevator control system (appear in Figure 8), a *Classification Category* class is redundant since the *Prioritize Request* feature was not selected[1]. The instantiation of each model should be valid

[1] Note that in the example only classes were classified by features, however, this can be done for each meta model element within the model (e.g., associations, attributes, and operations).

with respect to the multiplicity indicator defined within the domain model (in case the associated feature was selected). Furthermore, as allowed by the ADOM approach each view may be enriched with application specific model constructs.
Having set the application model, or at any time during its development a validation procedure may be executed in order to verify the consistency and completeness of the application model with respect to the domain model. The procedure includes the following stages:

1. Omitting application specific elements (i.e., the elements that are not classified as domain elements)
2. Checking for each instantiation of a feature whether an instantiation of the related view instantiations are created (as defined within the mapping model).
3. Checking that no redundancies exist in case of a feature was not selected but the related model elements were instantiated.
4. Checking the various instantiated views for their adherence with their corresponding domain views following the principles defined by the ADOM approach.

When utilizing the proposed approach we advocate that we provide the system designer specification guidelines and tools for managing the adherence of the model with the pre-defined constraints and for managing the integration and consistency among the various views.

4. Summary

In this paper we propose an approach for guiding system developers in specifying applications in multi-view environments. The approach is based on two domain engineering techniques, namely, feature modeling, and the Application-based Domain Modeling (ADOM) approach. We advocate that the integration of these techniques provides guidelines to system developers and a validation template for the entire system model. We set the procedure of working with the proposed approach and demonstrate its use via a case study.

We plan to further formalize the proposed approach and validation rules and to implement this within a CASE tool environment. In addition, we intend to conduct experiments to verify our conjecture regarding the extent to which the guidelines provided by the proposed approach help in achieving correct and complete specification of the desired applications.

Acknowledgement

This work was partially funded by Deutsche Telekom Laboratories at Ben-Gurion University.

References

1. Berenbach, B., Closing the Software Development Gaps with UML; *Proc., of the 16th Annual ACM SIGPLAN Conference on Object-Oriented Programming, Systems, Languages and Applications. OOPSLA 2001.* Tampa, Florida, 2001.
2. Czarnecki, K. and Antkiewicz, M., Mapping Features to Models: A Template Approach Based on Superimposed Variants. GPCE 2005, 422-437, 2005.
3. Czarnecki, K., Helsen, S.,and Eisenecker, U., Formalizing cardinality-based feature models and their specialization, Software Process: Improvement and Practice, 10 (1), 7-29, 2005.
4. Dobing, B. and Parsons, J., How UML is used, Communication of the ACM 49 (5) 2, 109-113, 2006.
5. Duffy, D. J., Domain Architectures: Models and Architectures for UML Applications. John Wiley & Sons, 2004.
6. Eisenecker, U. and Czarnecki, K. 2000. Generative Programming: Methods, Tools, and Applications. Addison-Wesley, Reading, MA.
7. Gomaa, H., Designing Software Product Lines with UML: From Use Cases to Pattern-Based Software Architectures. Addison-Wesley, 2004.
8. Hwan, C. , Kim, P., and Czarnecki, K., Synchronizing Cardinality-Based Feature Models and Their Specializations, Model Driven Architecture – Foundations and Applications, LNCS 3748, 331-348, 2005.
9. 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.
10. Lee, K., Kang, K. C., Koh, E., Chae, W., Kim, B., and Choi, B. W., Domain-oriented engineering of elevator control software: a product line practice. In *Proceedings of the First Conference on Software Product Lines: Experience and Research Directions: Experience and Research Directions* (Denver, Colorado, United States). P. Donohoe, Ed. Kluwer Academic Publishers, Norwell, MA, 3-22, 2000.
11. Peleg, M. and Dori, D., The model multiplicity problem: experimenting with real-time specification methods, *Software Engineering, IEEE Transactions on* 26(8), 742-759, 2000.
12. 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, 410-424, 2004.
13. Reinhartz-Berger, I. and Sturm, A., Enhancing UML Models: A Domain Analysis Approach, Journal of Database Management 19(1), 74-94, 2008
14. Reinhartz-Berger, I., Conceptual Modeling of Structure and Behavior with UML – The Top Level Object-Oriented Framework (TLOOF) Approach, ER'2005, LNCS 3716, 1-15, 2005.
15. Shoval, P. and Kabeli, J., FOOM-functional and object-oriented methodology for analysis and design of information systems. In *Advanced Topics in Database Research Vol. 1*, K. Siau, Ed. IGI Publishing, Hershey, PA, 58-86, 2002.
16. 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), Lecture Notes in Computer Science 3288, 766-779, 2004.