# Domain Aspects: Weaving Aspect Families to Domain-Specific Applications

Iris Reinhartz-Berger

Department of Management Information Systems,
University of Haifa, Haifa 31905, Israel
iris@mis.haifa.ac.il

**Abstract.** The exponential growth of data and information in the last decade has caused a rapid increase of system complexity. Two ways to face the emerging challenges are aspect-orientation and Software Product Line Engineering (SPLE). However, most of the works in these areas deal with specific aspects that are woven to concrete systems or product lines. Recent works suggest incorporating aspect-orientation to different tasks in software product line engineering, mainly variability specification and management. For improving reusability, validation, and compatibility of aspects, we suggest in this work recruiting an Application-based DOmain Modeling (ADOM) approach in order to define families of aspects and their weaving rules to families of applications during the entire development lifecycle. In particular, three types of models, namely aspect, base, and woven models, are defined in different abstraction levels and exemplified using UML notation.

**Keywords:** aspect-oriented modeling, early aspects, software product line engineering, domain analysis, metamodeling, UML

## 1 Introduction

The significant increase in systems complexity in the last decade has increased the need for software engineering techniques for dividing and decomposing complex problems into smaller ones, which may be solved one at a time with relatively simple means, and for gathering and composing these solutions into holistic solutions that solve the complex problems at hand. As long as the decomposed parts are orthogonal to each other, the separation of concerns can be easily achieved. However, when the concerns are interdependent, it is difficult or impossible to achieve completely separated parts. *Aspect-oriented software development* (AOSD) [8, 13] aims at providing modularization according to which crosscutting concerns are separated from traditional units during the entire software development lifecycle. Percolating aspect-orientation to early development phases [4], aspect-oriented modeling [25] deals with defining, specifying, weaving, and managing crosscutting concerns during the requirements, analysis, and design phases.

Focusing on representation and weaving of aspects at the modeling level, many works apply UML or its extensions (e.g., [1], [3], [9], [11], and [22]), while several

use goal-related notations (e.g., [16], [32]). However, most of these works deal with either aspects at a high level of abstraction, partially specifying (or completely neglecting) the weaving guidelines, or present aspects and weaving guidelines very close to implementation. Furthermore, these works mainly deal with general aspects that fit all systems and, hence, their weaving rules are quite vague, or with aspects which were particularly developed for, or integrated to, a specific system and, hence, are limited in their reusability capabilities. Several works, such as JPDD [26], define query models for specifying the models to which a specific aspect model can be woven. They further use these query models, along with parameters and templates, for defining weaving rules. This way join points may not be explicitly specified in the base systems (to which the aspects are woven).

Recent works have investigated the relationships between aspect-orientation and software product line engineering (SPLE), which is a field dealing with sets of software-intensive systems that share common, managed features satisfying the specific needs of particular market segments or missions [29]. Kuhlemann et al. [15] concluded that feature-orientation, a leading SPLE approach, is closely related to aspect-orientation. Furthermore, feature-oriented methods suffice in many situations where aspect-orientation is commonly used. Apel and Batory [2] have noticed that Aspect-Oriented Programming and Feature-Oriented Programming are complementary technologies: the weakness of one maps to the strength of the other. Different works use aspects for tackling various SPLE obstacles: implementing heterogeneous crosscuts [18], managing variability [19, 28], instantiating and customizing product line architectures [17], and so on. These works focus on weaving a particular aspect to a generic system that can be derived into different products in order to fulfill particular requirements. They do not support specifying and designing families of aspects and weaving rules that can be similarly applied to domains of applications or systems.

In this paper, an approach for defining families of aspects and their weaving rules to families of applications during the analysis and design phases is suggested. This approach utilizes the Application-based DOmain Modeling framework (ADOM), presented in [23, 24] in order to develop and maintain domain aspects and their weaving into particular applications. Aspect, base, and woven models are defined at different abstraction levels of ADOM; namely application and domain levels.

The main contribution of the work is two-fold. First, we enable designing and representing families of aspects together, capturing their commonality and variability. This way reusability, validation, and compatibility can be applied to aspects and not just to systems and software. Second, aspects can be woven to families of applications rather than to specific or generic applications. In particular, we offer specifying a match pattern as part of the aspect model. This match pattern defines the (minimal) requirements from the systems to which the aspect is intended to be woven. This way, the weaving rules can be more specific to the domain at hand and yet applied to different applications in that domain.

The remainder of this paper is organized as follows. Section 2 describes the ADOM approach, while Section 3 elaborates on its extension towards aspect-orientation. For demonstrating the approach, UML class diagrams are used on a case

study of a Check-In Check-Out (CICO) domain and a security aspect family[1]. Section 4 reviews related works, discussing their advantages and shortcomings with respect to the proposed approach. Finally, Section 5 concludes and refers to future research directions.

## 2 Application-based DOmain Modeling (ADOM)

The framework at the basis of the Application-based DOmain Modeling (ADOM) approach [23, 24] is comprised of three layers: application, domain, and language. The *application layer* consists of models of particular applications and systems, including their structure and behavior. The *language layer* includes metamodels of modeling languages, such as UML. The intermediate *domain layer* consists of specifications of various application families or product lines, including their common features and allowed variability. Furthermore, constraints among the different layers are enforced; in particular, the domain layer enforces constraints on the application layer, while the language layer enforces constraints on both the domain and application layers.

Separating the application and domain layers from the language layer, ADOM can be used in conjunction with different modeling languages, but when adopting ADOM with a specific modeling language, this language is used in both application and domain layers, easing the task of application creation (instantiation) and validation by employing the same constructs and terminology in both layers [24]. In this research we chose to apply ADOM to the widely used modeling language, UML [21], in order to create a workable dialect of ADOM, called ADOM-UML.

For expressing multiplicity-related constraints in the domain layer, a <<multiplicity>> stereotype is defined in the language layer. This stereotype is associated to the top level *Element* metaclass in order to represent how many times a model element of this type can appear in a specific context. The multiplicity stereotype has two associated tagged values, min and max, which respectively define the lowest and upper most multiplicity boundaries. For clarity purposes, four commonly used multiplicity groups are defined on top of this stereotype: <<optional many>>, where min=0 and max= ∞, <<optional single>>, where min=0 and max=1, <<mandatory many>>, where min=1 and max= ∞, and <<mandatory single>>, where min=max=1. Any multiplicity interval constraint can be specified using the general <<multiplicity min=$m_1$ max=$m_2$>> stereotype. The multiplicity stereotypes of dependent elements (i.e., elements that depend on other elements in the model, such as attributes that depend on their owning classes) and relational elements (i.e., elements that connect other elements such as associations between classes) are interpreted according to the "elements context". For example, defining an attribute of an optional class as mandatory means that each application class that instantiates the optional domain class must have this attribute. However, valid applications in the domain may have no instantiations of the class and its attributes.

[1] A complete version of this example, including application of the approach to UML 2.0 sequence diagrams, can be found at http://mis.haifa.ac.il/~iris/research/CICOexample.pdf.

A domain model in ADOM-UML includes the main concepts of the domain and the relations among them expressed in UML. In particular, the multiplicity stereotypes are used in the domain layer in order to specify cardinality-related commonality and variability, i.e., denote the range of possible instantiations of domain elements. In addition, ADOM employs the modeling language expressiveness and semantics in order to specify additional constraints in the domain layer. The metaclass of an element, for example, imposes constraints on its instantiation; namely domain classes can be instantiated by application classes, domain associations can be instantiated by application associations, and so on.

As an example of a domain model in ADOM-UML, consider Check-In Check-Out (CICO) applications [14] which manage operations for item enrollment and signing-out. Examples of applications in this domain include library, car rental, hotel management, and version control systems. The class diagram in Fig. 1 constrains the structure of such applications: any application in this domain must have at least one class of `Items`, at least one class of `Loaners`, and at least one class of `Lending` objects. The abilities to maintain a `Waiting List` in case the items are occupied and to handle `Item Types` (as sometimes reservation is done for item types rather than for individual items) are optional, as not all the applications in the domain support this functionality. The CICO domain model further specifies that each instantiated `Item` class has one attribute identifying the item ID, zero or more enumerated attributes denoting the item status, and zero or more descriptive attributes. Each `Item` class also exhibits at least one operation for getting the item details and possibly exhibits operations for getting and updating the item status. Requiring a queue implementation, each `Waiting List` class exhibits at least one operation for adding nodes to the list, at least one operation for getting the first node details, and at least one operation for removing the first node. As the `Waiting List` class is optional, these operations may not appear in a CICO application model. However, if such class is instantiated, these operations are mandatory.

Having a domain model, it is used as a reference for developing the target application model. The relationships between a generic element and its instantiations are maintained by domain classifiers, represented as stereotypes. In addition, some generic elements may be omitted and not included in the application model (these should be specified as optional elements in the domain model) and some new specific elements may be inserted to the specific application model (these are termed application-specific elements). Nevertheless, the domain knowledge embedded in the generic model must be maintained in the specific one. The class diagram in Fig. 2 exemplifies an application model in ADOM-UML which specifies a library system in the CICO domain. This application model contains two types of loaners (`Students` and `Staff` members), two types of item types (`Books` and `Multimedia`), one type of items (`Copies`), one type of waiting lists (book `Reservations`), and `Loans`. Note that since the different loaners have similarities in the application model, inheritance relations are used in the application layer. Furthermore, the library system model maintains the structure constrained by the CICO domain model, presented in Fig. 1. In this case, `Author` and `Student Reservation` are application-specific classes and all optional CICO domain classes are instantiated at least once.

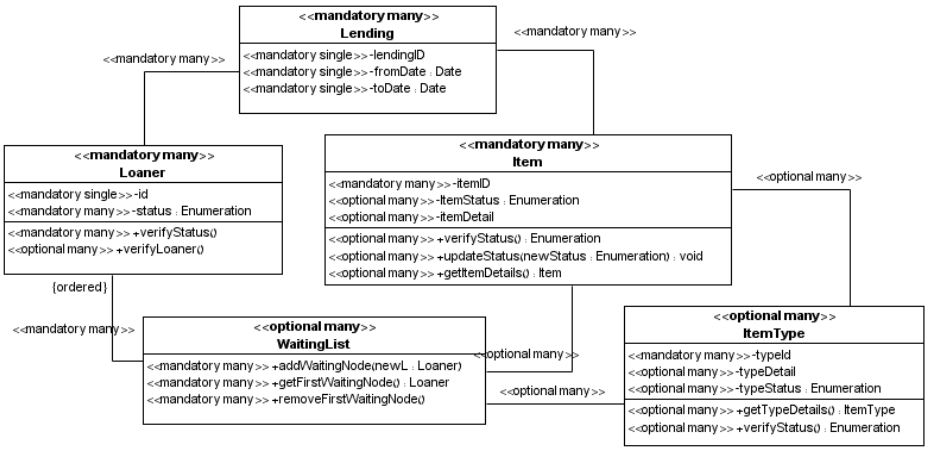


**Fig. 1.** A CICO domain model

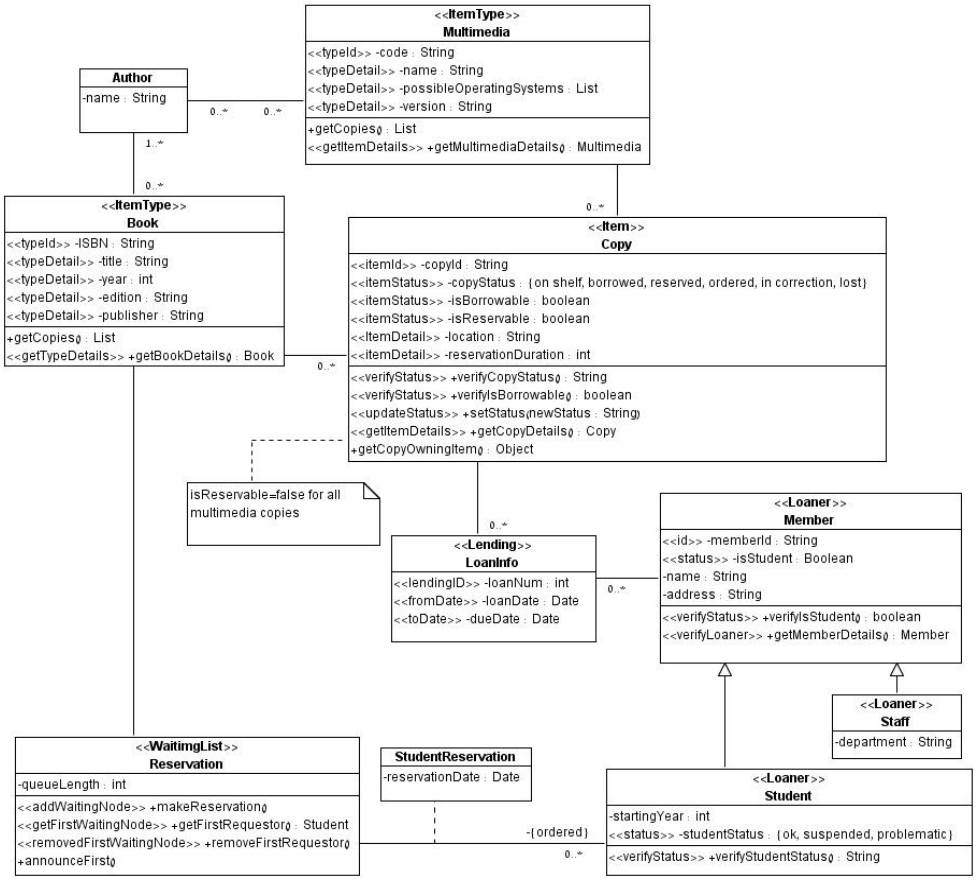


**Fig. 2.** A library application model

## 3 Aspect-Oriented ADOM-UML

The aspect-oriented ADOM-UML approach distinguishes among three types of models: base, aspect, and woven models. A *base model* describes an application, a system, or a family of such (i.e., a domain). Base models are described in ADOM-UML as explained and exemplified in Section 2. An *aspect model* describes a particular concern and its possible weaving to different systems. Similarly to [19], ADOM-UML aspect models can be described as triples of concern specifications (CS), match patterns (MP), and merge guidance (MG). The *concern specification* deals only with issues that are relevant to the concern at hand (*what* will be woven). The *match pattern* constrains the range of base models to which the aspect is applicable (*where* the aspect will be woven). Finally, the *merge guidance* specifies guidelines for weaving the given aspect to any applicable base model (*how* to weave the aspect). Note that although the same concern specification may have several pairs of suitable match pattern and merge guidance models, we consider an aspect model as the aforementioned triple. In other words, several aspect models may share the same concern specification with different match patterns (and consequently different merge guidance). The model formed after weaving the concern model using the merge guidance into a base model that satisfies the match pattern is termed a *woven model*.

Differently from [19], an ADOM-UML aspect model can be defined in the application or domain layer, respectively specifying a specific concern or a family of "similar" concerns. This way rules for weaving aspect families to domains of applications can be specified. Fig. 3 summarizes the main model types in the aspect-oriented ADOM-UML approach and the relations between them, while the rest of this section elaborates and exemplifies each type of model.

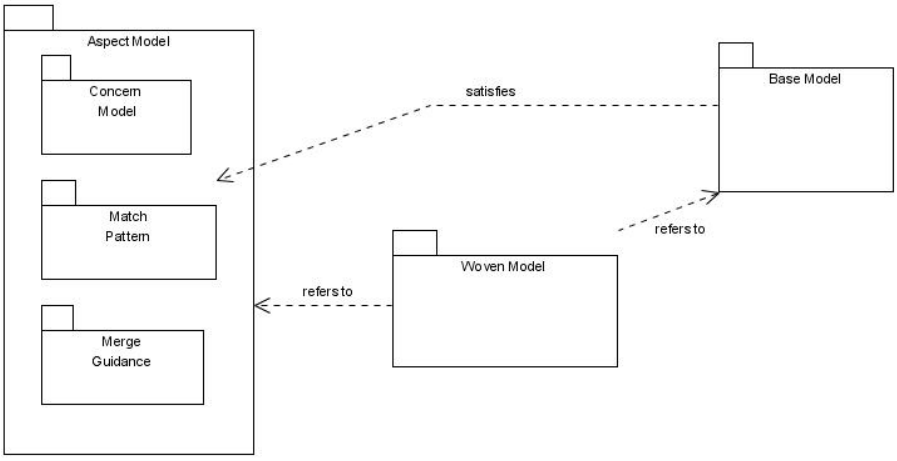


**Fig. 3.** A top-level view of the aspect-oriented ADOM-UML approach

### 3.1 Concern Specification

Being separated from the weaving rules, concern specifications are designed similarly to base models. As an example of a domain aspect (i.e., a family of aspects), consider security, which is a branch of computer science concerned with risk management trade-offs in the areas of confidentiality, integrity, and availability of electronic information [30]. Systems which contain fundamental flaws in their security designs cannot be made secure without compromising their usability. However, in many cases security techniques can be woven into (existing) system designs and, hence, considered as aspects. An example of a particular security aspect (which resides in the application layer) is authorization, which deals with protecting computer resources by allowing those resources to be used only by consumers that have been granted authority to use them. Other examples of particular security aspects are authentication and fraud protection.

The domain model depicted in Fig. 4 describes that each aspect in this security domain must deal with `Performers`, `Secured items`, and `Actions`. In addition, some of the applications in the domain may deal with `Policies` (referred to a specific performer, a specific item, or a specific performer-item pair) and record the `History` of security-related activities. Fig. 5 specified the particular authorization aspect, which enables executing only authorized actions by users: each `Item` is connected to `Users` through `Authorized Actions` and `Authorization` policies. Fig. 5 is an instantiation of Fig. 4, as all the domain level security constraints are satisfied in the authorization model. Note that `History` is not handled in this particular aspect.

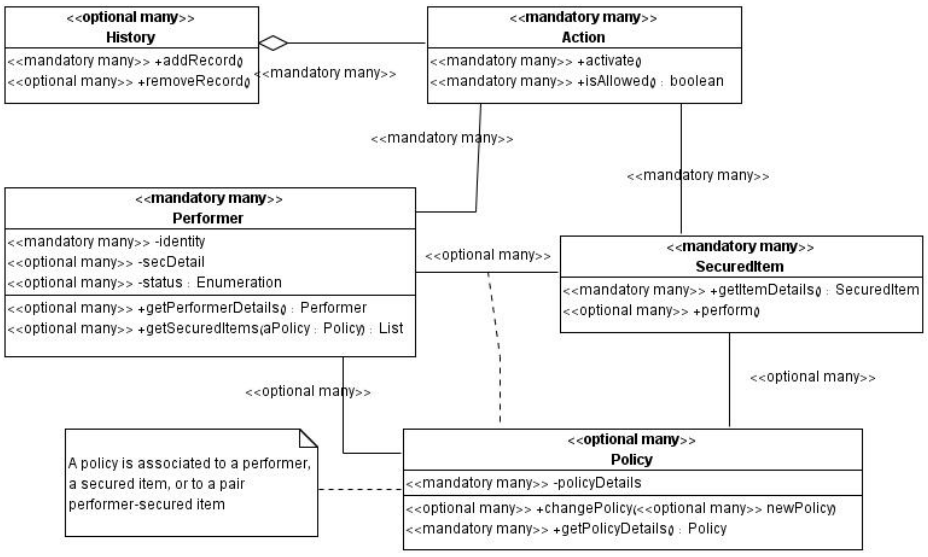


**Fig. 4.** A domain model of a security aspects family

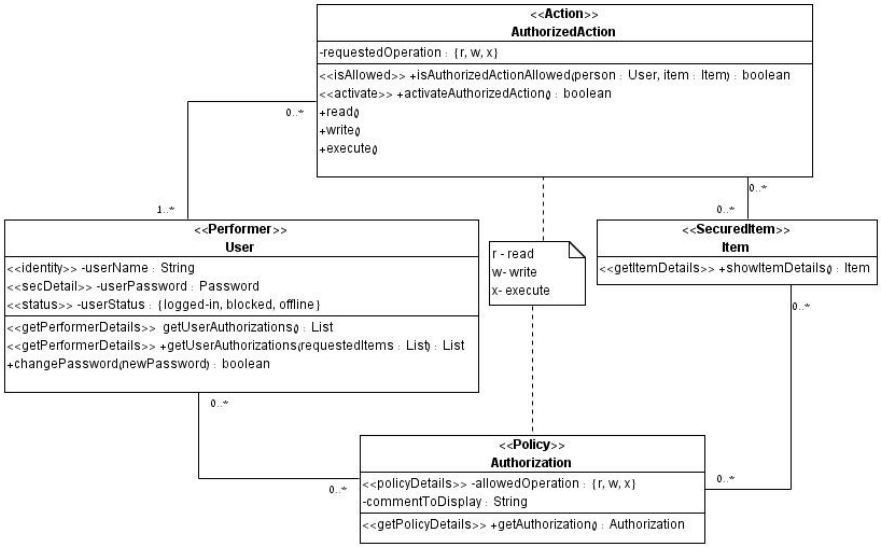


**Fig. 5.** An application model of an authorization aspect

### 3.2 Match Patterns

A match pattern is the part of an aspect model that specifies rules and constraints on the base models to which the concern specification can be woven. In other words, this part represents the minimal requirements from the base models that can weave the concern specification. Furthermore, as explained in the next section, a match pattern defines "anchors" (join points) to which the merge guidance can refer. The least restricting match pattern is the empty model which implies that the aspect model in general and its concern specification in particular are applicable and can be woven to any base model. Making the match pattern more detailed reduces the number of base models to which the concern specification can be woven, but enables specifying more reasonable and detailed weaving rules. The aim of match patterns is similar to that of Joint Point Designation Diagrams (JPDD) [26]: to specify all properties that a model element must provide in order to represent a join point. However, as opposed to JPDD that defines joint points on particular aspects and applications (base models), our approach enables definitions of match patterns at the domain layer, implying the specification of similar joint points to all the applications in the domain.

To visually specify match patterns in ADOM-UML, we define in the language layer a <<matchcond>> stereotype, which is associated to the top level *Element* metaclass in the UML metamodel and has two associated tagged values: elCardinality and elConstraint. elCardinality specifies the range of elements required to be matched to this element in the base model, whereas elConstraint constrains the possible

matched elements using OCL [20]. The default values of these tagged values (when not presented) are the less restricting ones, namely elConstraint is true and elCardinality is optional (0..n). As an example, Fig. 6 exemplifies a match pattern of the security domain aspect for domain base models. This match pattern requires that the base model to which the concern specification will be woven includes `Products` whose names syntactically or semantically contain the string 'product' and exhibit operations that return void. Furthermore, as the default value of elCardinality is 0..n, the base model may have `Controlling Elements`, each of which has a name that syntactically or semantically contains the string 'control' or 'system', a multiplicity stereotype of mandatory single or mandatory many, and may have association to `Product` elements. The CICO domain model specified in Fig. 1 satisfies the match pattern depicted in Fig. 6. In particular, `Item` corresponds to `Product`[2], while `updateStatus` corresponds to `operationOnProduct`. The other two operations of `Item` do not match `operationOnProduct`, since they do not return void. Similarly, `Item Type` does not match `Product`, since it does not have "void" operations. Furthermore, the security aspects family in general and the authorization aspect in particular can be woven to the library system, specified in Fig. 2, as this system handles updateable items in the form of copies (of multimedia or books).

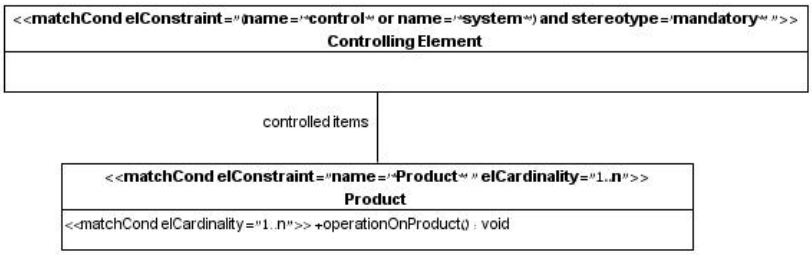


**Fig. 6.** The match pattern for the security aspect

### 3.3 Merge Guidance

The merge guidance of an aspect model combines the concern specification and the match pattern of the same aspect in order to guide the designer how to weave the concern specification into a base model that satisfies the match pattern. For this purpose, the elements of the match pattern and the concern specification are used as the elements of the merge guidance[3].

---

[2] Their linguistic similarity according to Wu and Palmer formula [31] is 0.7143 (out of 1), meaning that they are relatively close terms.

[3] If the name spaces of the concern specification and the match pattern of the same aspect are not distinctive, then adding the model (package) name to the element names is required.

We distinguish between four types of operations: combining, concern addition, merge addition, and match only operations. A *combining operation* takes two elements, one from the concern specification (CS) and the other from the match pattern (MP), and combines them into a third element that exhibits the features of the two composing elements. This third element appears in the merge guidance, where its name is taken from the match pattern and its stereotype – from the concern specification. A *concern addition operation* enables adding elements from the concern specification that does not exist neither have counterparts in the base model. These elements appear in the merge guidance without names and with stereotypes taken from the concern specification. A *merge addition operation* enables adding elements that do not appear in the concern specification neither in the match pattern, but are rather required when merging or weaving the concern specification into a base model that satisfies the match pattern. The names of these elements do not appear in the match pattern and their stereotypes are not taken from the concern specification. Finally, a *match only operation* enables specifying elements that are required only for matching base models, but are not modified as a result of weaving the concern specification into the base model. They identically appear in both match pattern and merge guidance.

As an example consider the concern specification depicted in Fig. 4 and the match pattern specified in Fig. 6. A possible merge guidance (MG) is depicted in Fig. 7[4]: `SecuredItem` from the concern specification is combined with `Product` from the match pattern (where `perform` is combined with `operationOnProduct`). `Controlling Element` identifies a match only operation. The merge guidance further adds associations between `Controlling Element` (if exists in the base model) and `Action` and `History` (from the concern specification). This merge guidance can be used in order to weave any security aspect (e.g., authorization) to any application in the CICO domain (e.g., the library system).

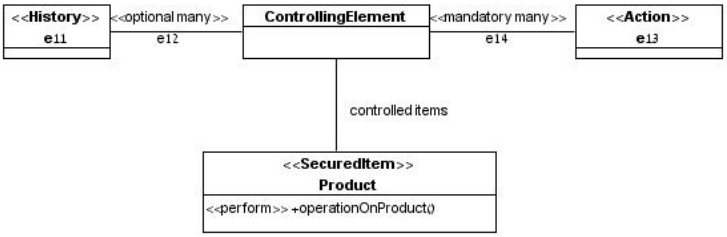


**Fig. 7.** The merge guidance of the security aspect family

A woven model is achieved by finding maximal matches between a base model and a match pattern and replacing each such occurrence with the merge guidance. Note that there may be more than one maximal match in a given base model that satisfies a single merge guidance element, implying application of the same merge rule several times to the base model (with different model portions). Furthermore, woven models can be created in the application or domain layers. In particular, merge

[4] The different $e_i$ are name replacers.

guidance models in the domain layer can help develop woven models in the application layer, which weave concrete aspects to particular applications. Fig. 8, for example, presents an authorized library system model that follows the security domain aspect in general and its merge guidance part in particular. In this figure, the elements that belong only to the base model are depicted in white, the base model elements that are combined with aspect elements are depicted in bold and grey, and the elements that are added due to the aspect are depicted in grey. The woven model is generated here only for the purpose of comprehending better the meaning of the aspect model parts. In the resultant woven model, the terminology (i.e., element names) is first taken from the base model and only afterwards (for additions) from the aspect model (or more accurately, the merge guidance).

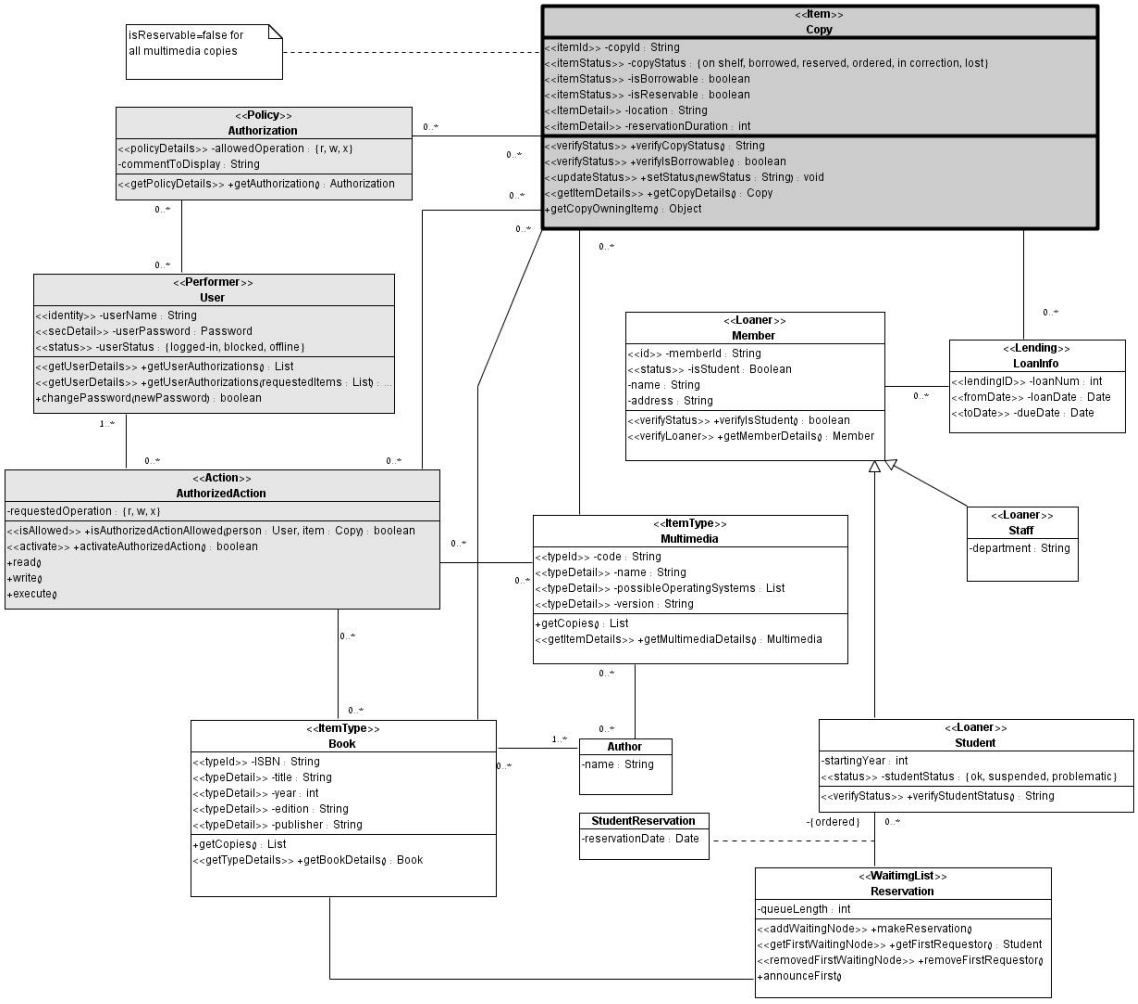


**Fig. 8.** An authorized library model resulted from weaving a security aspects family into the CICO domain

## 4 Related Work

Many works in early aspects refer to visual modeling in general and UML in particular. They usually introduce different sets of UML stereotypes or profiles for representing aspect-oriented design concepts (e.g., [1], [7], [9], and [27]). They deal with specific aspects, partially handling (or completely neglecting) weaving guidelines, or present aspects and weaving guidelines very closely to the implementation and programming level, falling short in considering the entire spectrum of modeling concepts not present in programming languages [25]. Baniassad and Clarke [3, 6], for example, suggest Theme/Doc and Theme/UML for aspect-oriented analysis and design. Theme/UML [7], which aims at modeling concerns, extends UML with two types of composition relations, merge and override, that involve an entire UML unit (e.g., attribute, method, or class). They do not explicitly refer to the conditions that a base model should fulfill so that the specific aspect will be woven into it. Furthermore, the weaving guidelines are specified as part of the aspect models, narrowing the ability to reuse the same aspect in different contexts.

Kande [12] proposes a Perspectival Concern-Space (PCS) technique for developing architecture with concerns as primary dimensions. Using UML, the PCS framework provides means for composing and decomposing different concern spaces. Groher and Voelter [10] present a model weaver that supports weaving of both models and metamodels. This weaver, which is developed on top of the Eclipse Modeling Framework [3], enables modeling optional parts as aspects and weaving them to a system at will. Reddy et al. [22] present a signature-based aspect composition approach. This approach refers to model element's signatures defined in terms of their syntactic properties, namely attributes or association ends. No separation between the concerns and the weaving rules is made in these works, reducing the ability to weave the same concern to different base models.

Stein et al. [26] suggest a special kind of diagram, called Joint Point Designation Diagrams (JPDD), for visualizing the selection criteria of base models to which an aspect can be woven. The notation extends the UML metamodel and is accompanied by a set of OCL operations for validating the selection queries on a modeling context. However, the approach does not support specifying weaving rules on the basis of these query models.

In the field of software product line engineering (SPLE), aspect-orientation is used for implementing heterogeneous crosscuts, managing variability, instantiating and customizing product line architectures. Lopez-Herrejon and Batory [18] suggest emulating function composition in aspect-oriented programming using a small set of advice, bounded quantification, and algebraic specification. Using ADORA, Stoiber et al. [28] propose visualizing and modeling variability using aspect-orientation and table-based modeling of configuration possibilities and constraints. Morin et al. [19] argue that aspect-oriented modeling can help users design optional and variant parts of a model. They further claim that the ability to weave aspects incrementally into base models enables constructing final products step-by-step. Their generic approach supports generating target languages and some weaving instructions to any given metamodel. After deriving an aspect by choosing the most appropriate variants and

options, aspect configurations can be woven into base models, to integrate new features and propose different variants of the system. Kulesza et al. [17] allow for improved customization and instantiation of frameworks by using crosscutting feature models. The approach intends to provide guidelines to modularize the implementation of framework features using aspects.

All these reviewed aspect-oriented SPLE approaches refer to aspects as particular concerns and, hence, suggest different ways to weave them to specific systems or product lines. The proposed ADOM-based approach enables modeling aspect, base, and woven models at different abstraction levels using the same modeling language. Moreover, the upper abstraction layer is used for reusing and guiding the development of models in the lower abstraction layer.

## 5 Summary and Future Work

Aspects refer to crosscutting concerns that should be woven to systems. Currently, aspects are specified either in a high level of abstraction or in the lowest level of design and implementation. The field of SPLE offers techniques and approaches for weaving aspects into software product lines rather than into particular applications (software products). In this work, we aim at enhancing the reusability of aspects by enabling specification of families of aspects and weaving rules. Adopting the Application-based DOmain Modeling (ADOM) framework, the proposed approach refers to three types of models; namely base, aspect, and woven models. An aspect model is further divided into three parts: (1) concern specification, which refers to issues of the aspect itself, (2) match pattern, which includes conditions on the base models to which the aspect can be woven, and (3) merge guidance, which comprises guidelines and rules for weaving the aspect to any base model that satisfies the match pattern conditions. The resultant woven models define the semantics of the different models and their related operations. Each model may reside at the domain or application layer of ADOM, respectively increasing or decreasing its level of generality. Four types of weaving processes can be defined on the basis of these models (see Table 1).

The proposed approach supports three main tasks in AOSD: aspect identification and representation (through the concern specification), join point determination (through the match patterns), and weaving guidelines definition (through the merge guidance). By separating an aspect into three different models, each type of model remains in a reasonable size, improving comprehensibility and maintainability. In addition, evolvability, which is the property of software that can easily be updated to fulfill new requirements [5], is enhanced: aspects depend on base models through match patterns. Therefore, they can be added or removed with minor changes to the different parts. Changes to the aspect itself, for example, may influence the concern specification and the merge guidance parts, while changes to the weaving process may influence the match pattern and the merge guidance parts.

Managing the different kinds of models at two different abstraction levels, the domain and application layers, enables generalizing families of models and instantiating them into valid application models. The domain models are used for

representing knowledge on families of models, including their weaving rules, enabling the reuse of this knowledge in particular cases and validating that the specifications of these cases do not violate the elicited knowledge of the corresponding domain. Furthermore, the expressiveness of the approach promotes its usability. It enables designing and representing families of aspects together, capturing their commonality and variability. This way reusability, validation, and compatibility can be applied on aspects and not just on systems and software. In addition, aspects can be woven to families of applications rather than to specific applications or generally to all the applications. Hence, the weaving rules can be more specific to the domain at hand and yet applied to different applications in that domain, enhancing once again the reusability of aspects.

We started developing a supporting tool for the aspect-oriented ADOM-UML approach. In its current state, the tool is plugged into an existing UML CASE tool (TOPCASED) and checks the correctness and completeness of specific base and aspect models (with respect to their base and aspect domain models)[5]. In the future, we intend to enhance the tool to automatically generate resultant woven models and check the consistency between the different parts of an aspect model (namely, concern specification, match pattern, and merge guidance).

**Table 1.** The meaning of weaving aspect and system models in different abstraction levels

| | **Level of aspect** | **Level of system** | **The meaning of the weaving** | **An Example** |
|---|---|---|---|---|
| a. | Domain | Domain | The result resides at the domain layer. Both system and aspect models should be instantiated into a particular system that includes specific aspects. | Weaving the Security aspect family into CICO applications |
| b. | Domain | Application | The result resides at the domain layer. The aspects that belong to the same family and are included in the particular system have to be instantiated. | Weaving the Security aspect family into the library system |
| c. | Application | Domain | The result resides at the domain layer. A family of system models includes the particular aspect. Each system in the family similarly integrates the particular aspect into its architecture. | Weaving the authorization aspect into CICO applications |
| d. | Application | Application | The result resides at the application layer. The particular system includes the particular aspect. No instantiation or further treatments are required. | Weaving the authorization aspect into the library system |

## References

1. Aldawud, O., Elrad, T. and Bader, A. A UML Profile for Aspect Oriented Modeling. Workshop on Advanced Separation of Concerns in Object-Oriented Systems, 2001.

[5] The tool can be downloaded from http://mis.hevra.haifa.ac.il/~iris/ADOM-UMLtool.zip.

2. Apel, S. and Batory, D. When to use features and aspects?: a case study. Proceedings of the 5th international conference on Generative programming and component engineering (GPCE'06), 2006, pp. 59-68.
3. Baniassad, E. and Clarke, S. Theme: An Approach for Aspect-Oriented Analysis and Design. Proceedings of the 26th International Conference on Software Engineering (ICSE'2004), IEEE Computer Society, 2004, pp. 158-167.
4. Chitchyan, R., Rashid, A., Sawyer, P., Garcia, A., Alarcon, M.P, Bakker, J., Tekinerdogan, B., Jackson, A., and Clarke, S. Survey of Aspect Oriented Analysis and Design Approaches. AOSD-Europe Network of Excellence, http://www.aosd-europe.net/, 2005.
5. Chris L., Rosenblum, D.S., and van der Hoek, A. The evolution of software evolvability. Proceedings of the 4th International Workshop on Principles of Software Evolution (IWPSE '01), 2001, pp. 134-137.
6. Clarke, S. and Baniassad, E. Aspect-Oriented Analysis and Design: The Theme Approach. The Addison-Wesley Object Technology Series, 2005.
7. Clarke, S. Extending standard UML with model composition semantics. Science of Computer Programming, 44 (1), Elsevier Science, 2002, pp. 71-100.
8. Elrad, T., Filman, R.E. and Bader A. Aspect-Oriented Programming: Introduction. Communications of the ACM, Vol. 44, No. 10, 2001, pp. 29-32.
9. Fuentes, L. and Sànchez, P. Designing and Weaving Aspect-Oriented Executable UML models. Journal of Object Technology, vol. 6, no. 7, Special Issue on Aspect-Oriented Modeling, 2007, pp. 109-136, http://www.jot.fm/issues/issue_2007_08/article5.
10. Groher, I. and Voelter, M. XWeave – Models and Aspects in Concert. Proceedings of the 10th Workshop on Aspect-Oriented Modeling, 2007.
11. Groher, I. and Baumgarth, T. Aspect-Orientation from Design to Code. Workshop on Early Aspects: Aspect-Oriented Requirements Engineering and Architecture Design, AODS'2004, 2004, http://trese.cs.utwente.nl/workshops/early-aspects-2004/Papers/Groher-Baumgarth.pdf.
12. Kande, M.M. Perspectival Concern-Spaces: An Aspect-Oriented Architectural Framework", Workshop on Aspect-Oriented Modeling with UML, 2002.
13. Kiczales, G., Lamping, J., Mendhekar, A., Maeda, C., Lopes, C., Loingtier, J. M., and Irwin, J. Aspect-Oriented Programming. Proceedings of the 11th European Conference on Object-Oriented Programming (ECOOP'97), LNCS 1241, 1997, pp. 220-242.
14. Kim, D., France, R., and Ghosh, S. A UML-Based Language for Specifying Domain-Specific Patterns. Journal of Visual Languages and Computing, Special Issue on Domain Modeling with Visual Languages 15(3-4), 2004, pp. 265-289.
15. Kuhlemann, M., Rosenmuller, M., Apel, S., and Leich, T. On the Duality of Aspect-Oriented and Feature-Oriented Design Patterns. Proceedings of the 6th workshop on Aspects, components, and patterns for infrastructure software (ACP4IS '07), 2007, pp. 5-11.
16. Kulesza, U., Garcia, A. and Lucena, C. Generating aspect-oriented agent architectures. Proceedings of the 3rd Workshop on Early Aspects, AOSD'2004, 2004.
17. Kulesza, U., Garcia, A., Bleasby, F., and Lucena, C. Instantiating and Customizing Product Line Architectures Using Aspects and Crosscutting Feature Models. Workshop on Early Aspects, OOPSLA'2005, 2005.
18. Lopez-Herrejon, R. and Batory, D. From Crosscutting Concerns to Product Lines: A Function Composition Approach. Technical Report TR-06-24, University of Texas at Austin, 2006.
19. Morin, B., Barais, O., and Jézéquel, J.M. Weaving Aspect Configurations for Managing System Variability. 2nd International Workshop on Variability Modelling of Software-intensive Systems (VaMoS'2008), 2008, pp. 53-62.
20. OMG: "UML 2.0 OCL Specification", (2004), http://www.omg.org/docs/ptc/03-10-14.pdf

21. OMG: "Unified Modeling Language: Superstructure", Version 2.0, (2005), http://www.omg.org/docs/formal/05-07-04.pdf
22. Reddy, Y. R., Ghosh, S., France, R., Straw, G., Bieman, J., McEachen, N., Song, E. and Georg, G. Directives for Composing Aspect-Oriented Design Class Models. Transactions on Aspect-Oriented Software Development, LNCS 3880, 2006, pp. 75-105
23. Reinhartz-Berger, I. and Sturm, A. Enhancing UML Models: A Domain Analysis Approach. Journal on Database Management (JDM), 19(1), special issue on UML Topics, 2008, pp. 74-94.
24. Reinhartz-Berger, I. and Sturm, A. Utilizing Domain Models for Application Design and Validation. Accepted for Information and Software Technology journal, 2009.
25. Schauerhuber, A., Schwinger, W., Kapsammer, E., Retschitzegger, W., Wimmer M., and Kappel, G. A Survey on Aspect-Oriented Modeling Approaches. 2006, Available at: http://www.bioinf.jku.at/publications/2006/1506.pdf
26. Stein, D., Hanenberg, S., and Unland R. Expressing different conceptual models of join point selections in aspect-oriented design. Proceedings of the 5th International Conference on Aspect-Oriented Software Development (AOSD'2006), 2006, pp. 15 - 26.
27. Stein, D., Hanenberg, S. and Unland, R. A UML-based Aspect-Oriented Design Notation for AspectJ. Proceeding of the 1st International Conference on Aspect-Oriented Software Development, ACM, 2002, pp. 106-112.
28. Stoiber, R., Meier, S., and Glinz, M. Visualizing Product Line Domain Variability by Aspect-Oriented Modeling, Proceedings of the 2nd International Workshop on Requirements Engineering Visualization (REV'2007), 2007, pp. 8-13.
29. Weiss, D. M. and Lai, C.T.R. Software Product-Line Engineering: A Family-Based Software Development Process. Addison-Wesley Professional, 1999.
30. Wikipedia: Security engineering, 2007, http://en.wikipedia.org/wiki/Security_engineering
31. Wu, Z. and Palmer, M. Verb Semantics and Lexical Selection. Proceedings of the 32nd Annual Meeting of the Association for Computational Linguistics, 1994, pp. 133-138.
32. Yu, Y., Leite, J. C. S. d. P. and Mylopoulos, J. From Goals to Aspects: Discovering Aspects from Requirements Goal Models. The 12th International Conference on Requirements Engineering (RE'2004), 2004, pp. 38-47.