

Specifying and Composing Concerns Expressed in Domain-Specific Modeling Languages [★]

Aram Hovsepyan, Stefan Van Baelen, Yolande Berbers, Wouter Joosen

Katholieke Universiteit Leuven, Departement Computerwetenschappen,
Celestijnenlaan 200A, B-3001 Leuven, Belgium
{Aram.Hovsepyan, Stefan.VanBaelen, Yolande.Berbers, Wouter.Joosen}@cs.kuleuven.be

Summary. Separation of concerns and levels of abstraction are key software engineering principles that can help master the increasing complexity of software applications. Aspect-oriented modeling (AOM) and domain-specific modeling languages (DSML) are two important and promising approaches in this context. However, little research is done to investigate the synergy between AOM and DSMLs. In this paper we present an asymmetric approach to compose modularized concerns expressed in different DSMLs with an application base model expressed in a general-purpose modeling language (GPML). This allows to specify each concern in the most appropriate modeling language. We introduce the concept of a concern interface, expressed in a GPML, that serves as a common language between a specific concern and the application base. In addition, we use an explicit composition model to specify the syntactic and the semantic links between entities from the different concerns. We explore these concepts using an application where we modularize the user interface modeled in WebML and the access control specified in XACML. The modularized concerns are then composed with an application base that has been specified in UML.

1 Introduction

The increasing complexity of software applications requires improved development techniques. The introduction of aspect-oriented modeling (AOM) and the evolution of domain-specific modeling languages (DSML) are very promising in this context. AOM [1] is a recent development paradigm that aims at providing support for separation of concerns at higher levels of abstraction by using Model-Driven Engineering (MDE) techniques [2]. The main goal of AOM approaches is to enable both vertical and horizontal separation of concerns [3].

In order to specify each concern, we have the possibility of using either a general purpose modeling language (GPML), such as the UML, or a DSML.

[★] The described work is part of the EUREKA-ITEA EVOLVE project, and is partially funded by the Flemish government institution IWT (Institute for the Promotion of Innovation by Science and Technology in Flanders), by the Interuniversity Attraction Poles Programme Belgian State, Belgian Science Policy, and by the Research Fund K.U.Leuven.

Most of the current AOM approaches [4, 5, 6, 7, 8, 9] use UML along with its extension mechanisms for expressing all concerns. However, DSMLs can potentially improve the current AOM methods for specifying concern models since the developer can use an optimal DSML for each of the concerns involved.

In this paper we analyze the problem of concern compositions when these are expressed in different DSMLs. We present a pragmatic AOM approach based on existing MDE building blocks and on the concept of a concern interface that we introduce. This approach allows the composition of a base concern expressed in a GPML with concerns specified in DSMLs, which do not necessarily conform to the same metamodel. We explore these ideas on a case study that modularizes two concern types: an essential part of security, i.e., access control expressed in XACML [10], and a web-based user interface expressed in WebML [11]. We further provide support for code generation towards an AO platform.

The paper is structured as follows. In section 2 we sketch the problem statement in detail. In section 3 we describe our asymmetric framework for modularizing and composing reusable concerns expressed in DSMLs. In section 4 we demonstrate a sample use of this framework for two different concern types and illustrate the concepts on a case study. In section 5 we evaluate our approach and discuss the possible alternative solutions. We present the related work in section 6. Finally, we conclude and give insights on our future work.

2 Problem Statement

Concerns are an important motivation for organizing and decomposing software into manageable and comprehensible parts [12]. We use the term *concern* to uniformly refer to what AOSD practitioners often call an *aspect concern* and a *base concern* [13]. The *base concern* typically represents the functional backbone of a given application, whereas different *aspect concerns* represent functional and non-functional modules that augment the core. In addition, we define the term *concern type* to refer to a certain concern domain, such as access control, user interface, transactions, etc.

2.1 Introduction

Modularity and abstraction are fundamental principles of any Aspect Oriented Software Development approach [14]. Both concepts are central in the current state-of-the-art AOM approaches [4, 5, 6, 7, 8, 9]. These approaches provide techniques to decompose a system by separating concerns into modules. In addition, they provide adequate means to model each concern on a high abstraction level and then refine it all the way to the implementation.

However, most of the current AOM approaches are based on the use of a GPML, i.e., UML with its extension mechanisms [15]. Even though UML is a well-known modeling language and can be used to specify almost any software system, it is not optimal and may lead to cumbersome and hardly usable models

[16]. DSMLs are said to greatly improve the comprehensibility for each concern domain by providing notations and constructs at the level of abstraction of the problem domain [16, 17]. DSMLs offer substantial gains in expressiveness and ease of use compared to GPMLs for the domain in question. In addition, DSMLs narrow the gap between the problem and implementation domain [2]. Using a GPML, one needs to describe the solution in domain terms, then map it to the GPML in question and finally transform it to code. DSMLs support the description of the solution directly in domain terms.

2.2 Concern Composition

Modularity must be complemented by composability. The various concerns need to relate to each other in a systematic and coherent fashion. One may further explore these relationships by performing the composition at various levels in the development cycle.

Figure 1 presents the actual problem statement where four different concerns are modeled using four different DSMLs from three different technical spaces. The notion of a technical (or technological) space is defined as a model management framework along with a set of tools that operate on the models that can be defined within the framework [18]. Ideally, all DSMLs should belong to the same technical space, however as different technical spaces offer complementary features in reality this is not the case. *Concern1* and *Concern2* conform to *DSML1* and *DSML2* defined in the Ecore technical space, *Concern3* conforms to *DSML3* defined by the XSD (XML) technology and *Concern4* conforms to a *DSML4* defined by yet another *Metamodel*. Bezivin et al. [19] have outlined the importance of bridging the technical spaces, rather than recreating artefacts throughout the different spaces.

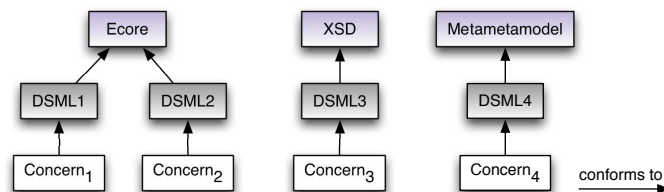


Fig. 1. Problem overview

Given the various concerns of the system expressed in different modeling languages across different technical spaces, we would like to specify and perform their composition in order to obtain a combined system. Note that throughout this paper we assume that suitable DSMLs for specifying modularized concerns are readily available.

In the next section, we present our approach for combining modularized concerns expressed in different modeling languages.

3 Concern Composition Framework

Once the modularized concerns are modeled, as illustrated in figure 1, one should be able to specify how the elements between different concerns relate to each other and to establish links between them. It is practically unmanageable to create one monolithic composition model to express all interrelationships between all concerns. This is why we will use pair-wise compositions between concern models. Only compositions between concerns related by a dependency relationship [20] should be specified. Dependency is a situation where one concern explicitly needs another concern and thus depends on it.

3.1 Asymmetric Approach using a GPML

We categorize the pair-wise concern compositions into two categories depending on the technical spaces each concern belongs to.

Compositions between concerns that lie in the *Ecore* technical space, i.e. intraspace compositions, are specified using AMW [21], which is an existing state-of-the-art framework for specifying model compositions. Each composition model conforms to a customized composition metamodel that extends the base abstract weaving metamodel provided by AMW. For instance, consider two concerns expressed in different DSMLs that conform to the *Ecore* metamodel (figure 2).

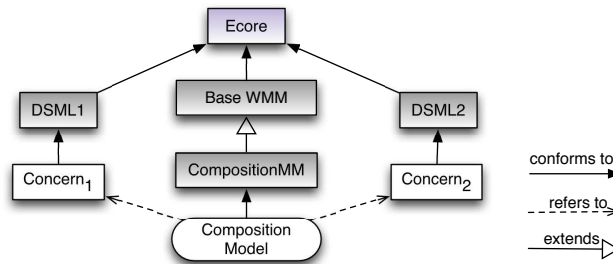


Fig. 2. Intraspace compositions with AMW

In order to specify their composition, one should extend the base abstract weaving metamodel and create a composition metamodel (*CompositionMM*) that is specific for *DSML1* and *DSML2*. One can then specify the composition between any two concerns expressed in *DSML1* and *DSML2* respectively.

For concerns that belong to different technical spaces, we will use the so-called interspace compositions. In order to specify an interspace composition we introduce a new concept called **concern interface**. A concern interface serves as a “lingua franca” (common language) between concerns expressed in different modeling languages. The use of a concern interface is illustrated in figure 3. In order to specify the composition between *Concern1* and *Concern2*, we create

a *ConcernInterface1* that represents the information required by *Concern1* expressed in *DSML2*. Concern interfaces are bridges between modeling languages from different technical spaces. In theory, concerns can be treated equally, i.e., we could also create a *ConcernInterface2* expressed in *DSML1* instead of *ConcernInterface1*.

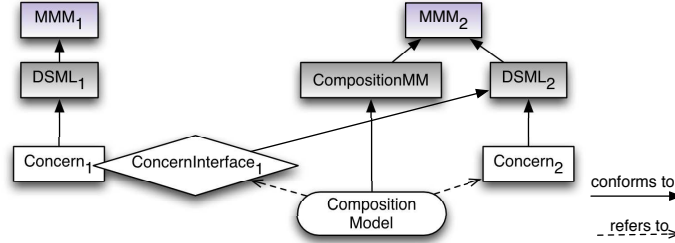


Fig. 3. Concern interface

As *ConcernInterface1* and *Concern2* are expressed in the same modeling language, we can use the intraspace composition specification methods. Unfortunately, this approach may not always work in practice. Even though DSMLs in general offer powerful abstraction mechanisms, they are limited to a specific domain. Hence, it could be impossible to specify *ConcernInterface1* in *DSML2*, because the latter does not offer the necessary concepts and constructs. The same holds for the inverse, where *ConcernInterface2* for *Concern2* should be specified in *DSML1*.

In order to solve this issue we introduce two constraints to the previous approach. First of all, we remove the symmetry from it. Even though, in theory, all concerns are equally important and should play symmetrical role [8, 22], an asymmetric approach is more practical. In an asymmetric approach one of the concerns, called a base concern, represents the functional backbone of the system. All other concerns are composed with this base concern. The second constraint is the usage a GPML for specifying the base concern. These restrictions ensure that it is mostly possible to create a concern interface. Finally, as the Ecore is one of the most advanced technical spaces in terms of tools we will use it as the base technical space.

Figure 4 shows the overall approach and the different composition types. *Concern1* is specified in *DSML1* that is native to *Ecore*, so it is composed with *Base* using an intraspace composition. *Concern2* and *Base* lie in different technical spaces and therefore we specify their composition using the interspace composition approach. We create a concern interface (*ConcernInterface2*) specified in *UML* and specify its composition with *Base* using the intraspace composition technique.

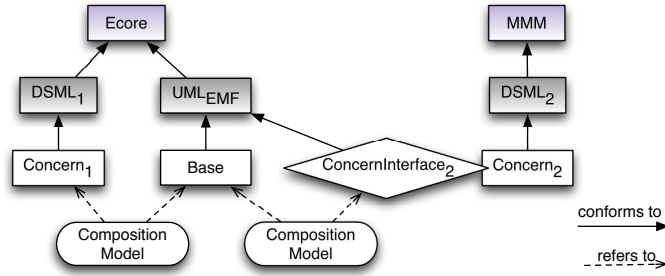


Fig. 4. The unified asymmetric solution

3.2 Composition Application

Once the composition pairs are specified, it is possible to perform each composition. There are two alternative solutions depending on the desired composition output. One possibility is to perform a model composition transformation and obtain a composed model. The composed model can then be further used to generate code for traditional OO platforms. The second possibility is translating each concern immediately into source code. In this case the target platform should take care of the composition itself and will typically be an AO platform. Note that in both solutions the composition application output will lose the domain specificity and be conform to a general purpose modeling or an AO programming language.

In this paper we use the latter alternative and transform each concern into AO source code. By targeting an AO platform, we can actually postpone the composition step towards the AOP level. The base concern model is transformed into a base code-level artefact. Each aspect concern model is translated into a code-level advice. Finally, the composition models are transformed into code-level pointcuts that specify how exactly the aspect concerns should augment the base code. The actual composition application is delegated to the AO code weaver that can completely automatically perform the composition of concerns and create a byte-level source code of the system.

In the next section we show an instantiation of our asymmetric approach for two different concern types. We illustrate both the intraspace and the interspace composition specifications and we use a model-to-code transformation tool to generate code for the CaesarJ [23] platform.

4 Case Study

In this section we explore the presented approach on a case study taken from the Electronic Health Information and Privacy (EHIP) domain.

4.1 Overview of the Method

First of all we present an overview of the method that we followed. Figure 5 presents an activity diagram showing the sequence of steps to be performed in order to develop an application by modularizing its access control model and its user interface model.

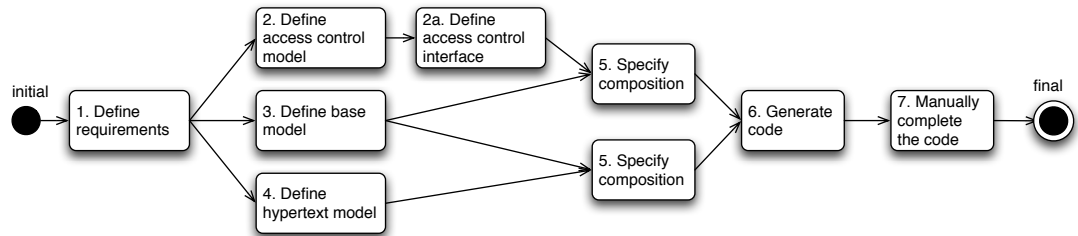


Fig. 5. Overview of the method

In the first activity the requirements are gathered and defined using existing approaches for requirements engineering. In activity 2 and 2a, the access control policies are written and an access control interface is created. In activity 3, the software engineer defines the base concern model. Activity 4 requires the specification of the user interface using a hypertext model. Activities 2, 3 and 4 can be done in parallel. In activity 5, the base-access control composition and base-user interface composition are specified. The last two activities involve semi-automatic code generation and its manual completion. Because of space restrictions we will not handle the first activity explicitly.

4.2 Conceptual Instantiation of the Concern Composition Framework

Our case study uses a screening lab application taken from the Electronic Health Information and Privacy (EHIP) domain. The application is an information management system in a screening lab and provides authorized access to relevant patient data. Figure 6 illustrates a high-level overview of two concerns applied on a screening application. We use UML as the GPML for the base concern model of the screening application. We specify the user interface of this application in WebML. We only use the WebML hypertext model in our case study. The WebML data model is actually completely defined by the base concern. A WebML implementation as an Ecore DSML is available, thus we can specify the composition with the base using the intraspace composition techniques. The access control is specified in XACML, which is a de facto standard language for specifying access control policies. As XACML lies in a different technical space, we also need to create the access control interface in UML in order to specify the composition with the base concern. The application of the composition specification targets the CaesarJ platform.

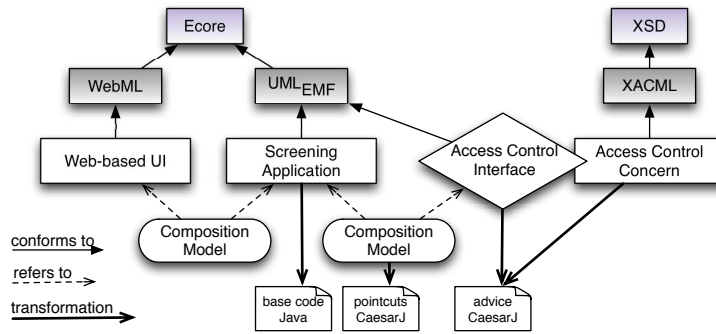


Fig. 6. Screening Application

Using the process described in the previous section we now illustrate the concepts from this figure on our case study.

4.3 Define Base Model

The screening lab application controls the information system of a screening lab. Figure 7 shows the structural base concern model expressed in UML. Patients (*ScreeningSubject*) make an appointment with a *Secretary* and have their radiographic pictures (*Screening*) taken by a *Radiographer*. Two different *Radiologists* perform a *Reading* of the radiographic screening. In case the reading results are the same an automatic *Conclusion* is generated. Otherwise a third reading takes place, where after a third radiologist creates a final conclusion.

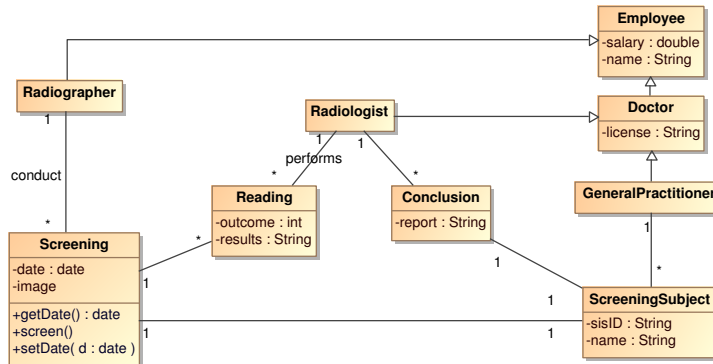


Fig. 7. ScreeningLab Application Model

4.4 Define Access Control Model

Many real world applications need to obey an access control policy in order to be deployed in practice. XACML is currently a de facto standard language in

practice that enables a relatively straightforward specification of security policies and its enforcement in applications. The access control policy for our application is largely defined by the general requirements in the health care domain.

A central concept in the health care domain is *contact*, which is essentially a workflow. A contact is a medical logical unit that can be an outpatient visit, a hospitalization, a surgery, a chemotherapy treatment, etc. A contact may result in a report after going through the following phases:

- reception: the patient and the department is associated with the contact;
- assignment: the contact is assigned to a physician and a supervisor;
- report generation: the assigned physician generates the report, which is then placed on the work list of the supervisor for validation;
- report validation: the assigned supervisor validates the report. This also closes the contact.

The assigned physician may append and modify relevant patient data. The assigned physician may also close the contact by formulating an explicit statement that no report is necessary. External general practitioners (GP) are able to view their patients' hospital records, so that they are kept informed of the health condition of their patients.

The data that needs to be protected encompasses the contact and all the reports that are associated with the contact. We will refer to all these documents as patient data. The access control policy that has to be enforced, consists of the following rules: (1) Physicians who are assigned to a contact are granted access to patient data related to that contact, until 30 days after the contact was closed, (2) a general practitioner retains his access rights as long as he remains registered as the patient's general practitioner, and (3) a physician can overrule an access denial, provided that a detailed reason is specified.

We specify these access control policies in XACML. Figure 8 shows a snippet of an XACML policy stating that a general practitioner may view his patient's medical record (some XACML specifics have been omitted for readability purposes). Each XACML policy or Rule has a feature called a Target, which is a set of simplified conditions for the Subject, Resource and Action that must be met for a Rule to apply to a given request. In our case **general practitioner** is the Subject, patient's medical record (**ehip:med:record**) is the Resource and **view** is the Action. Each Subject, Resource and Action is identified using Attributes, which are central concepts in XACML. Attribute values are resolved from a request using AttributeDesignators that specify an attribute with a given name and type. For instance, to identify the Subject of our policy we use the AttributeDesignator with the name *urn:e-hip:access-subject:attributes:role* and type *http://www.w3.org/2001/XMLSchema#string*. Once the Target is matched the Condition is evaluated. Each Condition is a boolean function. In our policy we use a standard XACML function *string-equal* with two Attributes the *subject-id* of the patient and the *gp-id* of the general practitioner. If the Condition evaluates to true then the Rule's Effect is returned (**Permit**). If the Condition evaluates to false, the Condition does not apply and NotApplicable is returned.

```

<Rule Effect="Permit" RuleId="urn:e-hip:ruleid:1">
<Target><Subjects><Subject>
  <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <SubjectAttributeDesignator AttributeId="urn:e-hip:access-subject:attributes:role"/>
    <AttributeValue>general practitioner</AttributeValue>
  </SubjectMatch></Subject></Subjects>
<Resources><Resource>
  <ResourceMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <ResourceAttributeDesignator AttributeId="urn:e-hip:resource:target-namespace"/>
    <AttributeValue>ehip:med:record</AttributeValue>
  </ResourceMatch></Resource></Resources>
<Actions><Action>
  <ActionMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
    <ActionAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:action:action-id"/>
    <AttributeValue>view</AttributeValue></ActionMatch></Action></Actions></Target>
<Condition FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-equal">
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
    <SubjectAttributeDesignator AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"/>
  </Apply>
  <Apply FunctionId="urn:oasis:names:tc:xacml:1.0:function:string-one-and-only">
    <ResourceAttributeDesignator AttributeId="urn:e-hip:resource:gp-id" />
  </Apply></Condition></Rule>

```

Fig. 8. XACML policy sample

Due to space restrictions we will not present the full XACML policies for our application ².

4.5 Define Access Control Interface

As the access control concern is expressed in a DSML from a technical space different from Ecore, we define an **access control interface** that serves as a bridge between the access control and the base concern. The access control interface consists of *object interfaces* that declare the available resources and a number of *subject interfaces* that declare the available subject roles. Both interfaces declare information that the policy may need, in the form of a number of attribute declarations. In addition, the object interface declares semantic actions. These semantic actions represent the security sensitive operations to which policies apply. All interfaces, attribute declarations and semantic actions are an abstract representation of the *Attributes* needed for the XACML policies from the previous subsection. In order to discern between security objects and subjects, we have created the $\ll object \gg$ and $\ll subject \gg$ stereotypes.

The first step towards creating an access control interface for the EHIP domain is filtering out security objects and security subjects from the organizational requirements and access control policies. We have defined four security subjects: *Doctor* and its specializations *GP*, *ResponsiblePhysician* and *Supervisor*; and one security object: *MedicalData*. The next step is finding the relevant security actions, which will be executed on the security objects. We have selected 4 security actions on *MedicalData* relevant to the case study: *view*, *append*, *close*

² The complete design models, sources, transformation and code generation templates and other relevant data used in this paper can be found at <http://www.cs.kuleuven.be/~aram/index.php?page=implementation1.html>

and *validate*. Finally, we have to specify the necessary security attributes for both security subjects and objects. For the *Doctor* security subject we have selected one relevant attribute - *licenseID* - which uniquely identifies each doctor in general. *GP* and *Supervisor* introduce no additional attributes. *ResponsiblePhysician* has a *reason* attribute for the overrule of an access denial (policy rule 3). In addition it has a *supervisor* attribute to denote the supervisor. *MedicalData* has a *patientID* attribute that determines the patient. It has a *closed* and *closingTime* attributes that denote whether the contact has been closed, and if so the exact closing date and time. Finally, *gp* and *responsiblePhysician* attributes are used to specify the license ID for the general practitioner and responsible physician for the contact. Fig. 9 shows the access control interface for our application that we have created from the access control concern model.

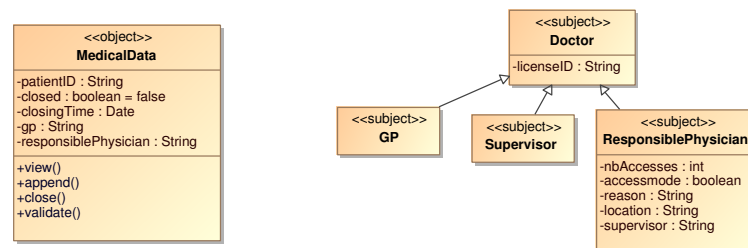


Fig. 9. EHIP Access Control Interface

4.6 Define Hypertext Model

We have specified the user interface concern using WebML. Figure 10 illustrates a partial hypertext model of the application. The initial page (Screening Home) contains a menu that can be used to navigate to the linked pages. The *Patients* page shows an indexed list of patients and is further navigable to view each patient's detailed information including the conclusion of the screening procedures. The *Screenings* page contains an index of screenings that are to be read by the radiologists. The users can then navigate to view each screening detail and the readings associated with it.

4.7 Specify Base-Access Control Composition Model

We need to map the concepts from the screening application, which are relevant for access control, to the access control interface. Based on the code-level solution by Verhanneman et al. [24, 25] we have created a composition metamodel that can be used to specify any composition between an application and its access control interface. The composition metamodel, shown in figure 11 is implemented as an extension of the generic weaving metamodel specified by AMW

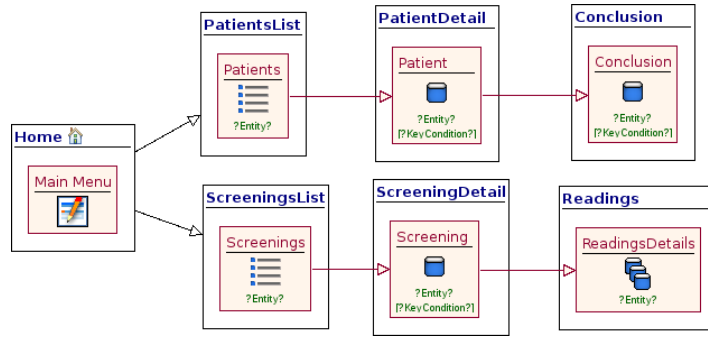


Fig. 10. User interface model

[21]. *MapObject* and *MapSubject* link types extend the *WLink* type from the AMW metamodel and are used to map classes from the base concern to security objects and subjects from the access control interface. *SecurityObject*, *SecuritySubject* and *BaseClass* are link end types that extend the *WLinkEnd* type. *MapAction* link is used to link security sensitive operations from the base concern with the corresponding security actions from the access control interface. Finally, *MapAttribute* links are used in order to specify the necessary attribute calculation used to perform the access control.

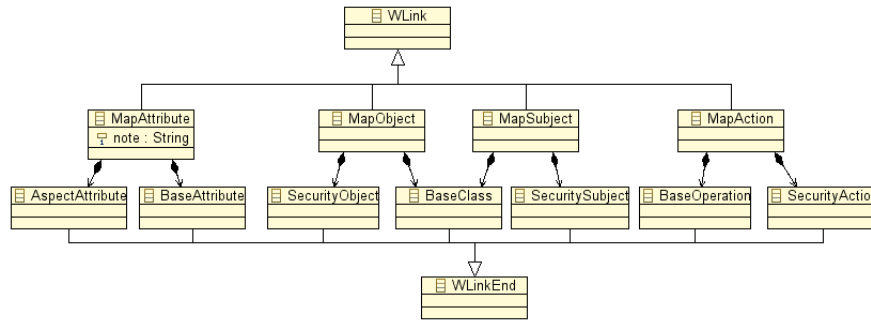


Fig. 11. Composition Metamodel

As AMW weaving tool has limited visualization capabilities, figure 12 shows only a partial snapshot of the access control composition model for the ScreeningLab application. The highlighted mapping links Screening class from the base with the MedicalData security object from the access control interface.

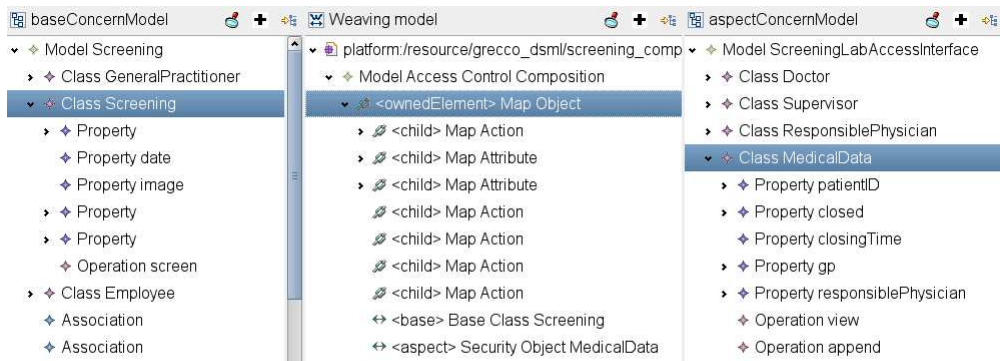


Fig. 12. Composition Model ScreeningLab

4.8 Specify Base-User Interface Composition Model

We have created another composition metamodel based on AMW base abstract weaving metamodel in order to specify the composition of the WebML model with the base concern. We have defined two types of composition links. The first type maps the WebML model entities (*Entity*) to the base concern. The second type defines the key condition that guides the how the relevant entities are selected. The entity mappings from the WebML model to the base are rather straightforward. The entities on *PatientsList* and *PatientDetail* pages map to *ScreeningSubject* class from the base, entities on *Conclusion* to *Conclusion*, entities on *ScreeningsList* and *ScreeningDetail* to *Screening*, and the ones on *Readings* to *Reading*.

4.9 Generate and Manually Complete Code

We have created a model-to-code transformation engine using MOFScript [26] that uses the base concern model, the XACML policies, its concern interface and the composition model as inputs and produces source code for the CaesarJ platform as an output. Currently, only the base-access control composition is realized in the generator. For a more detailed description of the generation engines, we refer to [27].

5 Discussion and Evaluation

In this paper we have identified two different concern composition categories based on the notion of a technical space: intraspace and interspace. The concept of modular reasoning is important throughout all technical spaces as technical spaces already provide the fundamentals and the toolset of concern modularization and composition using different modeling languages within the technical space. We have used these building blocks and we have demonstrated the feasibility of both intraspace and interspace compositions using a GPML for the

base concern and DSMLs for the aspect concerns. However, we have introduced a number of restrictions in our pragmatic approach. In this section we provide an analysis of possible alternatives given the problems presented in section 2 and evaluate our approach.

5.1 Intraspace Compositions

There are two possible solutions in order to specify the composition between concerns that belong to the same technical space [16].

One possible solution is to integrate the metamodels of the concern models by the means of shared or linked modeling constructs. This approach is very similar to the way the UML metamodel combines the different types of diagrams and information. For instance, the entity concept can be used on one hand to specify objects in a sequence diagram model and on the other hand to specify the classes in a structural data model. The advantage of this approach is that the concern composition is already specified at the meta-level. However, such an integration is very impractical especially if the involved metamodels are complex and evolving. Moreover, we would need to integrate every pair of DSMLs for which a composition specification is necessary.

A second option is to use a special composition model that specifies the syntactic and semantic mapping between the elements of the involved concerns. Currently, many technical spaces already provide techniques and tools for composition specification. The XML technical space supports schema modularization and composition out of the box, ATLAS Model Weaver (AMW) [21] is a generic model composition framework for the Eclipse Modeling Framework (EMF) technical space, AspectJ [28] could be used as one of the modularization and composition mechanisms for the Java technical space, etc.

In our approach we have demonstrated the feasibility of the second option. Our approach is constrained to an asymmetric setting where the base concern is specified using a GPML. However, these constraints are not essential for the intraspace composition specification, but are introduced only for the interspace composition specifications.

Intraspace composition application is also supported by the technical spaces, which provide powerful transformation tools. Tools like ATLAS Transformation Language (ATL) [29] for Ecore, XSLT and XPath for XSD, and aspect weaver for AspectJ are capable of handling the composition specifications and transforming them into composed models. However, given two concerns that belong to different DSMLs it remains unclear what their composed model should look like. Obviously, one would need to provide an intelligent combination of the input metamodels in order to enable the intracomposition application. One of the alternative solutions is using a GPML as an output metamodel rather than trying to calculate the output DSML. In order to do so, besides the composition engine, one would also need a mapping scheme for each DSML concept onto a GPML. Another approach is generating code for an AO platform where the aspect weaver will ultimately compose the models at the code level. This approach, however, will not produce an integrated design view that could be used,

e.g., to better understand the interactions across the composed design views. Currently, we have not yet implemented the intraspace composition application of the screening lab from the case study.

5.2 Interspace Compositions

In theory, it is possible to create similar solutions for the interspace composition specifications as for the intraspace ones. The first approach would require one to integrate two metamodels from different technical spaces. Obviously, this option will be even more complex than that for intraspace composition specification. The second alternative would mean defining a more generic base abstract weaving metamodel that allows the specification of links between metamodels from two technical spaces. However, this solution would be quite complex as well. First of all, one would need abstract weaving metamodels for each pair of technical spaces. Although the number of technical spaces used will probably be rather limited, creating such a metamodel is a daunting task on its own as it has to conform to a sophisticated combination of the metamodels of the two technical spaces. Moreover, one would also need to create the tools that are capable of analyzing and transforming such interspace composition specifications.

An different approach presented by Bezivin et al. [19] proposes the concept of a projector, which is a special sort of transformation that can transform a concern from one technical space to another. A number of such projectors are already available to translate concerns between technical spaces (e.g. XMI, JMI). As the DSMLs we are using are typically small, they would require relatively little effort to develop a projector. One of the advantages of this approach is that all concerns can be transformed to the same technical space and eventually be treated equally. Hence, a symmetrical AOM approach could be feasible. However, this approach may be less practical in certain cases. For instance, if applied to our case study, we would need to translate each XACML policy to the base technical space and specify an equal number of compositions.

In this paper we have proposed a pragmatic approach that relies on the concept of a **concern interface**, which serves as a bridge between concerns expressed in different modeling languages. In this case the GPML use is essential as we need a language that is expressive enough to specify the concern interface. We have explored the interspace compositions by illustrating how one can compose access control concern expressed in XACML and base concern expressed in GPML. Initially, UML seems a good choice as it is expressive enough and we expect that one could specify any concern interface in UML. As EMF implements only the structural part of the UML standard, it might be impossible to create a concern interface for, e.g., a behavioral concern. However, we see this as a limitation of EMF rather than UML. Even though the Ecore technical space is central in our case study, it is not essential for the ideas presented.

There are several possible solutions in order to apply the interspace composition specification. The first alternative is to provide an intelligent combination of the input metamodels as a target output metamodel. However, as mentioned previously, this is quite complex to realize in practice. The second possibility is

to use the projector approach for the composition specification and rely on the intraspace composition application techniques. Finally, it is also possible to use the code generating approach like in the case of intraspace compositions.

The approach presented in this paper uses the last alternative, for which the input concerns are transformed into AO code where the real composition is done by the aspect weaver. Given the power of the AO programming languages, we assume that any model-based composition can be translated into an appropriate pointcut code-level construct. In [27] we have illustrated some initial insights on how targeting an AO platform may offer benefits over the traditional OO platforms in a given AOM approach. A number of pure code-level approaches have also shown that AO source code may improve certain software qualities (e.g. [30, 31]). Obviously, this approach will not work if OO source code is required. Moreover, there are no means to analyze concern interactions in order to identify for instance conflicts and undesirable emergent behaviors between concerns.

A potential drawback of our approach is that instantiations for each concern type are very specific and require a substantial effort. As we have illustrated in the previous section, the instantiation for the access control concern is completely different from the instantiation for the user interface concern. The cost of building a concern interface and a composition model may also be a disadvantage compared to using an integrated non-AOM based approach. For instance, one could use the SecureUML [32] approach in order to specify the access control policies immediately on the base concern models.

6 Related Work

To our best knowledge, there is no existing AOM approach that can specify and perform the composition of concerns expressed in different modeling languages from different technical spaces. However, there are a number of related approaches that we describe according to several categories. The first category outlines the relationships to the code-level AO approaches that implement the combination of a general purpose programming language (GPL) and a domain-specific programming language (DSL). The second category investigates the relationship with the most prominent AOM approaches. Finally, the last category presents the most prominent works in the area of DSMLs, access control modeling and data-intensive web application development in the context of MDE.

6.1 Domain Specific AOP Languages

Recent research on the combination of AO DSLs in order to tackle the increasing complexity and size of software applications has produced a number of interesting approaches for different domains [33, 34, 35]. The AO DSLs typically specify a crosscutting concern using an expressive DSL, while the rest of the system

is written in Java, which is a GPL. These approaches come with a low-level composition engine that can perform the code-weaving and produce plain Java byte-level code. The composition specification, i.e. the pointcuts, are written in the DSL as well. The intraspace compositions in our approach are conceptually very similar to these code-level solutions. However, in the code-level solution the base is typically unaware or oblivious [36] to the aspect. Because of the fragile pointcut problem [37] where changes to the base may break the aspect, uncontrolled use of the powerful pointcut constructs may require the base to deal with aspect details. This will lead to actually reduced modularity. In our approach, this problem is substantially reduced by using explicit links in the composition model rather than using a pattern matching mechanism offered by the pointcut language.

6.2 AOM Approaches

There are many GPML-based AOM approaches, each pursuing a distinguished goal, providing different concepts as well as notations [1]. UML is often used to specify the *base concern*. For *aspect concerns*, the existing approaches typically use a light-weight UML extension with the concepts specific for each approach. Typically most of the current approaches [4, 5, 6, 7, 8, 9], whether symmetric or not, allow the modularization and composition of concerns in order to obtain a combined view. All these approaches are intraspace compositions in our terminology where all concerns are expressed in a GPML, i.e., UML. These approaches are complementary to our approach when it comes to concerns that are better expressed using a GPML rather than a DSML, e.g., concerns that belong to the solution domain rather than the problem domain (e.g. design patterns [38]), or concerns for which no suitable DSML exists and creating one would be too costly.

All these approaches are in theory generic and can be used to model any concern type using UML. Song et al. [39] have demonstrated the use of a role-based access control (RBAC) within the AOM approach [4]. However, the approach where XACML can be used as it is to specify the access control policies is more suitable.

Aspect-Oriented Domain Modeling (AODM) of Gray et al. [40] is the most prominent AOM approach that explicitly focuses on the usage of DSMLs. AODM is an extension of Model-Integrated Computing [41] with a model weaver framework called the Constraint-Specification Aspect Weaver. Using AODM we can model an application using a DSML created by the Generic Modeling Environment tool and weave it with domain constraints written in the Embedded Constraint Language. However, AODM is limited to only one type of concerns, i.e. constraints. Our approach is complementary to AODM.

6.3 Other Approaches

A number of pure MDE approaches have successfully used DSMLs for specifying the software system on a higher abstraction level [16]. Even though these

approaches are very practical, concern modularization is only a desired option and is not yet achieved.

A number of approaches define UML extensions to provide support for access control. SecureUML [32] is the most prominent approach that uses extended RBAC with authorization constraints specified in OCL. However, SecureUML does not offer any mechanisms for specifying conflict resolution strategies, whereas defining them in XACML is straightforward. In addition, there is no support for policy grouping in SecureUML. Grouping of policies improves the performance of the system, as we can easily exclude certain policy groups from the authorization checks. Both conflict resolution and policy grouping are crucial in structuring and managing policies in complex applications. Finally, XACML offers support for obligations in policies. Moreover, even though we have used XACML, our approach is easily extensible towards other access control models and modeling languages based on RBAC. The only thing that would need to be reworked is the model transformation engine.

Ceri et al. [11] define an MDE framework that focuses on the design and semi-automatic implementation of data intensive web applications. The web-based user interface concern that we have illustrated is similar to their approach. However, our framework targets the modularization of multiple heterogeneous concerns and targets any kind of software systems.

7 Conclusions and Future Work

In this paper we have introduced the notion of a concern interface to bridge different technical spaces. We have explored a combined intra- and interspace composition framework that is built on existing MDE approaches and tools. We have successfully illustrated the usage of this framework on an application with modularized access control concern and web-based user interface concern. In the future, we plan to further investigate this composition framework for other DSMLs and perform a detailed cost/benefit analysis of using a combination of different modeling languages.

References

1. Schauerhuber, A., Schwinger, W., Kapsammer, E., Retschitzegger, W., Wimmer, M., Kappel, G.: A survey on aspect-oriented modeling approaches. Technical report (2006)
2. France, R., Rumpe, B.: Model-driven development of complex software: A research roadmap. In: FOSE '07: 2007 Future of Software Engineering, Washington, DC, USA, IEEE Computer Society (2007) 37–54
3. Simmonds, D., Reddy, R., France, R., Ghosh, S., Solberg, A.: An aspect oriented model driven framework. In: EDOC '05, Washington, DC, USA, IEEE Computer Society (2005) 119–130

4. Reddy, Y.R., Ghosh, S., France, R.B., Straw, G., Bieman, J.M., McEachen, N., Song, E., Georg, G.: Directives for composing aspect-oriented design class models. (2006) 75–105
5. Baniassad, E., Clarke, S.: Aspect-Oriented Analysis and Design: The Theme Approach. Addison-Wesley (2005)
6. Hanenberg, S., Stein, D., Unland, R.: From aspect-oriented design to aspect-oriented programs: tool-supported translation of jpdds into code. In: AOSD '07: Proceedings of the 6th international conference on Aspect-oriented software development, New York, NY, USA, ACM (2007) 49–62
7. Sanchez, P., Fuentes, L.: Designing and weaving aspect-oriented executable uml models. *Journal of Object Technology* (6) 2007
8. Jézéquel, J.M.: Model driven design and aspect weaving. *Software and Systems Modeling* (2008)
9. Hovsepian, A., Van Baelen, S., Berbers, Y., Joosen, W.: Generic reusable concern compositions. In: Fourth European Conference on Model Driven Architecture Foundations and Applications. (2008) 231–245
10. OASIS: Core specification: Extensible access control markup language (XACML) v2.0. (www.oasis-open.org/committees/xacml)
11. Ceri, S., Fraternali, P., Bongio, A.: Web modeling language (webml): a modeling language for designing web sites. In: Proceedings of the 9th international World Wide Web conference on Computer networks : the international journal of computer and telecommunications netowrking, Amsterdam, The Netherlands, The Netherlands, North-Holland Publishing Co. (2000) 137–157
12. Ossher, H., Tarr, P.: Multi-Dimensional Separation of Concerns and The Hyperspace Approach. In: Proceedings of the Symposium on Software Architectures and Component Technology: The State of the Art in Software Development. (2000)
13. Kiczales, G.: Aspect-oriented programming. (*ACM Comput. Surv.*) 154
14. Rashid, A., Moreira, A.: Domain models are not aspect free. In: In MODELS, Springer (2006) 155–169
15. OMG: UML superstructure, v2.0. OMG Document number formal/05-07-04 (2005)
16. Kelly, S., Tolvanen, J.P.: Domain-Specific Modeling: Enabling Full Code Generation. Wiley-IEEE Computer Society Press (2008)
17. Mernik, M., Hering, J., Sloane, A.M.: When and how to develop domain-specific languages. *ACM Comput. Surv.* **37** (2005) 316–344
18. Kurtev, I., Bezivin, J., Aksit, M.: Technological spaces: An initial appraisal. In: CoopIS, DOA'2002 Federated Conferences, Industrial track. (2002)
19. Bezivin, J., Kurtev, I.: Model-based technology integration with the technical space concept. In: Metainformatics Symposium 2005, Esbjerg, Denmark (2005)
20. Sanen, F., Truyen, E., Joosen, W.: Managing concern interactions in middleware. In: DAIS. (2007) 267–283
21. M. D. Del Fabro, J. Bezivin, P. Valduriez: Weaving models with the eclipse amw plugin. In: Eclipse Modeling Symposium, Eclipse Summit Europe 2006. (2006)
22. Moreira, A., Rashid, A., Araujo, J.: Multi-dimensional separation of concerns in requirements engineering. In: RE '05: Proceedings of the 13th IEEE International Conference on Requirements Engineering, Washington, DC, USA, IEEE Computer Society (2005) 285–296
23. I. Aracic, V. Gasiunas, M.M.K.O.: An overview of CaesarJ. In: Transactions on Aspect-Oriented Software Development. (2006) 135–173
24. Verhanneman, T., Piessens, F., De Win, B., Truyen, E., Joosen, W.: A modular access control service for supporting application-specific policies. Volume 7., Piscataway, NJ, USA, IEEE Educational Activities Department (2006) 1

25. Verhanneman, T., Piessens, F., De Win, B., Joosen, W.: Uniform application-level access control enforcement of organizationwide policies. In: ACSAC '05: Proceedings of the 21st Annual Computer Security Applications Conference, Washington, DC, USA, IEEE Computer Society (2005) 431–440
26. SINTEF: MOFScript. (<http://modelbased.net/mofscript/>)
27. Hovsepian, A., Van Baelen, S., Yskout, K., Berbers, Y., Joosen, W.: Composing application models and security models: on the value of aspect-oriented technologies. In: Proc. of the 11th Int. Workshop on AOM@MODELS'07. (2007)
28. G. Kiczales, E. Hilsdale, J. Hugunin, M. Kersten, J. Palm, and W. G. Griswold: An overview of AspectJ. Volume 2072. (2001) 327–355
29. Jouault, F., Kurtev, I.: Transforming models with ATL. In: Satellite Events at the MoDELS 2005 Conference, LNCS 3844, Springer (2006) 128–138 ISBN=0302-9743.
30. A. Garcia, C. Sant'Anna, E. Figueiredo, U. Kulesza, C. Lucena, A. von Staa: Modularizing design patterns with aspects: A quantitative study. In: 4th international conference on AOSD. (2005)
31. Greenwood, P., Bartolomei, T., Figueiredo, E., Garcia, A., Cacho, N., Sant'Anna, C., Borba, P., Uirakulesza, Rashid, A.: On the impact of aspectual decompositions on design stability: An empirical study. In: Proceedings of ECOOP 2007. LNCS, Springer-Verlag (2007) 176–200
32. Basin, D., Doser, J., Lodderstedt, T.: Model driven security: from uml models to access control infrastructures. *ACM Transactions on Software Engineering and Methodology* **15** (2006) 39–91
33. Fabry, J., Eric Tanter, DHondt, T.: Kala: Kernel aspect language for advanced transactions. *Sci. Comput. Program.* **71** (2008) 165–180
34. Amor, M., Garcia, A., Fuentes, L.: Agol: An aspect-oriented domain-specific language for mas. In: EARLYASPECTS '07: Proceedings of the Early Aspects at ICSE, Washington, DC, USA, IEEE Computer Society (2007) 4
35. Dinkelaker, T., Mezini, M.: Dynamically linked domain-specific extensions for advice languages. In: DSAL '08: Proceedings of the 2008 AOSD workshop on Domain-specific aspect languages, New York, NY, USA, ACM (2008) 1–7
36. Filman, R., Friedman, D.: Aspect-oriented programming is quantification and obliviousness. In: Workshop on Advanced Separation of Concerns, OOPSLA 2000, Minneapolis. (2000)
37. Sullivan, K., Griswold, W.G., Song, Y., Cai, Y., Shonle, M., Tewari, N., Rajan, H.: Information hiding interfaces for aspect-oriented design. In: ESEC/FSE-13: Proceedings of the 10th European software engineering conference held jointly with 13th ACM SIGSOFT international symposium on Foundations of software engineering, New York, NY, USA, ACM (2005) 166–175
38. E. Gamma, R. Helm, R. Johnson, J. Vlissides: *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional (1995)
39. Song, E., Reddy, R., France, R., Ray, I., Georg, G., Alexander, R.: Verifiable composition of access control and application features. In: SACMAT '05: Proceedings of the tenth ACM symposium on Access control models and technologies, New York, NY, USA, ACM (2005) 120–129
40. Gray, J., Bapty, T., Neema, S., Schmidt, D.C., Gokhale, A., Natarajan, B.: An approach for supporting aspect-oriented domain modeling. In: GPCE '03: Proceedings of the 2nd international conference on Generative programming and component engineering, New York, NY, USA, Springer-Verlag New York, Inc. (2003) 151–168
41. Sztipanovits, J., Karsai, G.: Model-integrated computing. *IEEE Computer* (1997)