Towards Semantic-Based Aspect Interaction Detection

Gunter Mussbacher[1], Jon Whittle[2], Daniel Amyot[1]

[1] SITE, University of Ottawa, 800 King Edward, Ottawa, ON, K1N 6N5, Canada
{gunterm, damyot}@site.uottawa.ca
[2] Dept. of Computing, InfoLab21, Lancaster University, Bailrigg, Lancaster, LA1 4YW, UK
whittle@comp.lancs.ac.uk

Abstract. Interactions between dependent or conflicting aspects are a well-known problem with aspect-oriented development. These interactions are potentially dangerous and can lead to unexpected or incorrect results when aspects are composed. To date, there have been very few attempts to address this issue at the modeling level. We present a new approach for detecting interactions that is based on lightweight semantic annotations of aspect models. Each aspect model is annotated with domain-specific markers and a separate goal model describes how semantic markers from different domains influence each other. When aspect models are composed, the composed model is inspected for any semantic markers that potentially conflict. This is achieved by propagating values through the goal model to see which goals (typically non-functional properties) are satisficed by the composition and which are not. The technique can be used both to highlight potential aspect conflicts and to trade-off aspects. We illustrate the approach using two aspect techniques.

Keywords: Aspect Interaction, Goal Models, Aspect-Oriented Modeling

1 Introduction

A well-known problem with aspect-oriented modeling (and also with aspect-oriented programming) is that multiple aspects may be applicable at a given point in the base model. This is known as the aspect interaction problem. In the best case, aspects may simply be ordered so that they are applied in an order which respects their dependencies. In the worst case, there may be deep semantic conflicts between aspects that require a rethinking of which aspects should be applied or require a remodeling of the aspects themselves. An example of the former is where an aspect assumes certain modeling elements in the base that must be introduced by another aspect. An example of the latter is a conflict between two non-functional aspects where there are inherent trade-offs that arise irrespective of the order in which the aspects are applied. A security aspect, for example, will inevitably affect a performance aspect because security mechanisms must be enforced. Conversely, a performance aspect may affect a security aspect – if, for example, the performance aspect caches results, which must then be protected. In such a case, it is not simply a matter of ordering the aspects in a certain way.

The aspect interaction problem is still largely unsolved[1]. One approach has been to explicitly document aspect interactions [2, 3]. This is useful but does not offer automated help in detecting interactions. Formal methods (e.g., model checking [4] or static analysis [5, 6]) have been applied to detect interactions, usually at the programming level. In previous work, we applied critical pair analysis to detect structural interactions between aspect models [7]. These techniques work well for interactions that can be detected by examining syntactic elements alone. They cannot handle cases that require a semantic interpretation of the model elements.

In this paper, we present a new approach for automatically detecting aspect interactions based on a lightweight semantic interpretation of model elements. Each aspect model is (manually) tagged with domain-specific *semantic markers* that define an interpretation of relevant model elements. In a distribution aspect model, for example, a server might be stereotyped as <<*local*>> or <<*remote*>>. In a security aspect model, a server that stores sensitive data might be stereotyped as <<*confidential*>>. When multiple aspects are applied to a model, certain model elements may end up with conflicting semantic markers. Applying both a distribution and a security aspect, for instance, could leave a server labeled both as <<*confidential*>> and <<*remote*>>. This is potentially a problem because confidential data sent across a network to a remote server must therefore be appropriately secured, whereas, for confidential data stored on a local server, the security implications are more modest.

Our technique relies on a set of domain-specific annotations for each aspect domain, as well as a model of how annotations from different domains influence each other. The latter is required for automatically analyzing interactions when multiple aspects are applied. In this paper, we represent influences between domain annotations as a goal model [8, 9]. This allows us to leverage existing analysis techniques for goal models to reason about trade-offs between conflicting aspects. Although the approach is generic to any kind of model, we illustrate it here with two different scenario modeling languages – aspect-oriented use case maps [9, 10] and aspect-oriented UML sequence diagrams [11].

The paper is structured as follows. Section 2 presents an example, which will be used throughout the paper as an illustration. Section 3 motivates the problem of semantically detecting aspect interactions. Section 4 presents the new approach. Section 5 describes related work and is followed by conclusions in Section 6.

2 Example: Electronic Voting

The focus in this paper is on models used for scenario-based requirements analysis. The underlying concepts, however, are applicable to any modeling language with a well-defined meta-model.

As an illustrative example, we will use a scenario model of an electronic voting machine (EVS) based on a description of Diebold's EVS in [12]. Diebold's EVS has two principal actor roles: voters who cast ballots, and poll officials who carry out a

[1] and is, of course, highly related to the feature interaction problem [1]

series of administrative tasks such as collecting the votes from a voting machine and reporting them to the voting authority. One of the principal use cases is the Reporting use case in which a responsible poll official interacts with the EVS to transmit the voting results to a central backend server. The Reporting use case will serve as the base model in this paper and is shown in Fig. 1 as a UML sequence diagram and as a use case map. (We use both use case maps and sequence diagrams to illustrate the broad applicability of the approach, but the paper can, in fact, be read focusing on one notation alone. In figures, sequence diagrams will be presented on the left and use case maps on the right.)

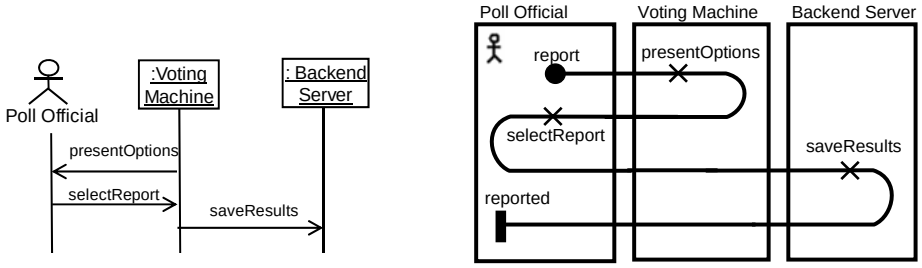

Fig. 1. Reporting use case.

We will not dwell on the details of the notations but will explain syntax as it arises. On the right-hand side (RHS) of Fig. 1, a reporting scenario is given by a use case map, which starts with a solid dot and ends with a solid bar. Crosses denote responsibilities attached to components. For example, the Voting Machine is responsible for presenting reporting options.

We will now consider how to apply three key aspects to the base model in Fig. 1:

- **Smart-card based authentication aspect**, which authenticates an actor before allowing him/her to carry out actions by providing authorized users with a smart card and PIN.
- **Remote Service**, which models provision of a remote service.
- **Caching**, which models a caching proxy on a local server.

These three aspects will be modeled as generally as possible and then applied to the base model so that: (1) polling officials are authenticated; (2) authentication is done by querying a remote database server; (3) authentication results are cached locally (since a polling official may need to access an EVS multiple times during the course of a voting session).

Fig. 2 shows aspect-oriented sequence diagrams and use case maps for the three aspects. The notation used for sequence diagram aspects is that of the MATA aspect modeling tool [11]. For use case maps, AoUCM (Aspect-oriented Use Case Maps) [13] is used. Both notations follow similar principles for modeling aspects. For each, an aspect is made up of two parts: a *pattern* to match against in the base model and an *aspect* which defines how the pattern is modified (e.g., by adding messages before or after messages in the base or by adding responsibilities to the base). In both cases, patterns may contain variables that may match any element in the base. Variables may be *atomic variables* that match a single element in the base or *sequence variables* that match a sequence of elements in the base.

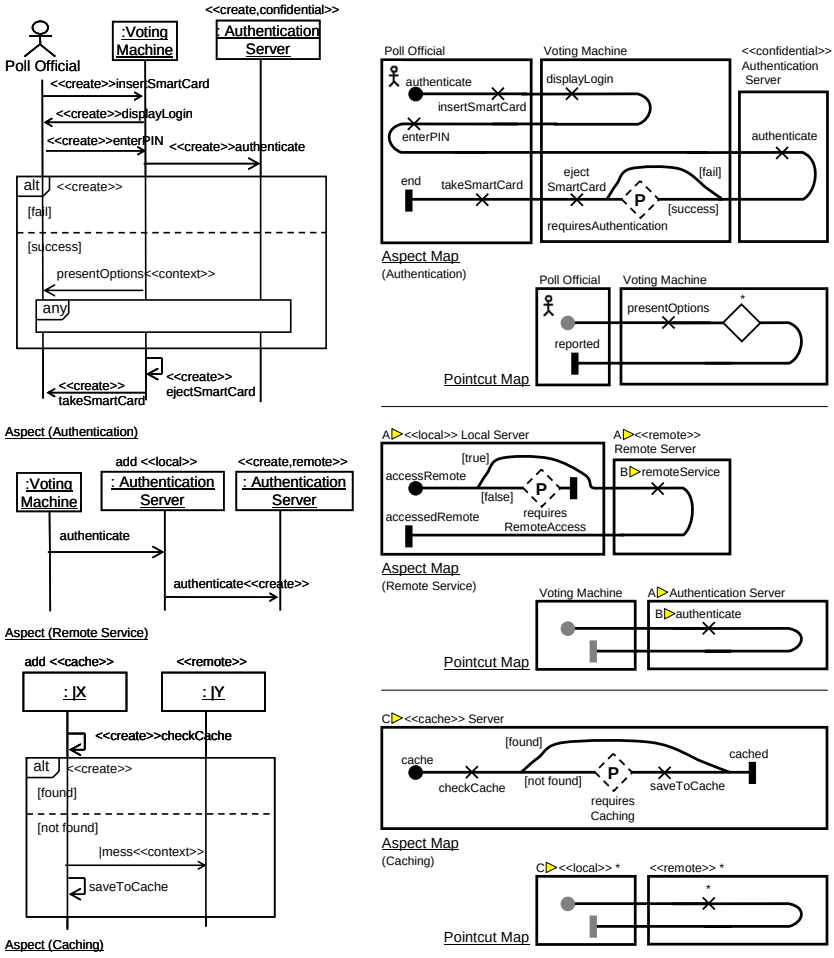

Fig. 2. Authentication, remote service, and caching aspects.

MATA makes use of a simple UML profile consisting of stereotypes <<*create*>>, <<*delete*>>, and <<*context*>>. The *pattern* consists of any elements without a stereotype or stereotyped with <<*delete*>> or <<*context*>>. The *aspect* consists of elements stereotyped with <<*create*>>. The aspect creates elements marked with <<*create*>> and removes elements marked with <<*delete*>>. Note that for container elements, such as interaction fragments, if <<*create*>> is applied to the container, it is by default applied to all contained elements unless a contained element is marked with <<*context*>>. Atomic variables in MATA are preceded by a vertical bar '|' and sequence variables are given by a special interaction fragment with operator **any** which matches against a sequence of messages. As an example, in Fig. 2, the Remote Service aspect matches against an authenticate message from the voting machine to

an object of type Authentication Server. The effect of the aspect is to add the <<*local*>> tag to the server, create a new server tagged with <<*remote*>>, and forward the message to the <<*remote*>> server. In the Authentication aspect in Fig. 2, the pattern is any sequence of messages beginning with presentOptions. The effect of the aspect is to wrap a new **alt** fragment around this sequence, to add a new authentication server and to add new authentication messages. The Caching aspect matches against any message sent to a <<*remote*>> object. The aspect converts the sending object into a cache and adds messages for dealing with local caching.

The principles for aspect-oriented use case maps are similar. *Patterns* are defined by *pointcut maps* which visually describe elements to match against in a base UCM model. Elements on a pointcut map may be variables. The *aspect* is defined by an *aspect map*. *Pointcut stubs* (represented by a dashed diamond) in an aspect map act as placeholders for any matches of the pattern found in the base UCM. The causal relationship of the *pointcut stub* and the aspect's properties visually defines how the aspect needs to be composed with the UCM model. Hence, in the top right of Fig. 2, the Authentication aspect matches against a path in the base consisting of presentOptions followed by any path of unspecified length (where the asterisk denotes any path and is hence a sequence variable). Composing this aspect with the base would result in a UCM that does an authentication check before options are presented.

The Remote Service aspect UCM in Fig. 2 generically represents a replacement of the matched pattern with behavior on the true branch which is always executed and invokes a remote service. The pointcut stub is placed on the false branch, indicating that it is never executed when the aspect is applied. The aspect does not specify what the remoteService is, since the aspect models the remote service in a generic way. It is the pointcut map that applies it to the authentication service. URN links (shown as small triangles in Fig. 2) allow an aspect to reuse matched behavior or structure from the pointcut map. For example, authenticate corresponds to remoteService and Authentication Server corresponds both to the local and remote server.

The Caching aspect applies to any interaction between a local and remote server and introduces a check against the cache and saving to the cache. A URN link is used to indicate that the local server is also the caching server.

Note that the models in Fig. 2 include domain-specific markers (<<*local*>>, <<*remote*>>, <<*confidential*>>, etc.). These will be used in our approach to give semantic meaning to model elements. We expect such markers to come from a pre-defined ontology or profile relevant to a particular domain.

3 Automatically Detecting Aspect Interactions

We distinguish between *syntactic* and *semantic* aspect interactions. Syntactic interactions can be detected by comparing the syntactic structures of two aspects to see if they overlap. Semantic interactions, on the other hand, require a deeper analysis of the semantics of model elements across aspects. Syntactic aspect detection is relatively easy to implement but cannot detect all kinds of interactions.

In previous work [7], we applied critical pair analysis (CPA), a graph transformation analysis technique, to detect syntactic interactions between MATA aspects. Since MATA aspects are graph rules, CPA can be used to examine overlaps between rules. An overlap corresponds either to: a dependency – one aspect requires a model element only introduced by another aspect; or a conflict – one aspect modifies the base in such a way that another aspect can no longer be applied.

Conflicts and dependencies usually imply that the rules should be applied in a particular order since the result may be different depending on which aspect is applied first. A conflict may also mean that two rules that should both be applied cannot be, and therefore, the rules themselves should be modified. The MATA tool applies CPA to automatically provide feedback on dependencies and conflicts.

As an example, consider the Remote Service and Authentication aspects from Fig. 2. There is a clear dependency between these two aspects since Remote Service needs to match in the base against an interaction between the Voting Machine and an authentication server. Since the Authentication aspect introduces this interaction, it must be applied before Remote Service. That is, there needs to be a local service before it can be converted to a remote service.

CPA is limited to detecting *syntactic* interactions because it is based solely on an analysis of the patterns used to define where the aspects match. For example, CPA can tell the modeler that Authentication should be applied before Remote Service, but there remains a *semantic* conflict between these two aspects *even if they are applied in the correct order*. This is because Remote Service sends data over a network. By itself, this presents no issues. However, when combined with the Authentication aspect, the data that is transmitted becomes confidential data, resulting in the transmission of sensitive data over a potentially insecure channel. There is therefore a conflict. The solution would be, of course, to secure the network, which would require an additional Encryption aspect. Our aim in this paper is to be able to flag such semantic conflict situations automatically.

Intuitively, our technique works by composing the model and looking for composed elements with more than one domain-specific marker. In this example, the composed model would contain a server marked as both <<*confidential*>> and <<*remote*>>. Given a separate semantic model that describes potential conflicts between these markers, the modeler can be notified of a possible problem. This simple idea is what we formalize in the remainder of the paper.

4 Semantic-Based Interaction Detection

4.1 Outline of the Approach

Our technique relies on a set of domain-specific *semantic markers* for each aspect domain, as well as a goal model of how semantic markers from different domains influence each other. The overall approach is illustrated in Fig. 3. First, a set of aspect models is defined (either in MATA or AoUCM) as well as a base model with which these aspects will be composed. Semantic markers are used to annotate each aspect

model, thus adding semantic meaning. The semantic markers are implemented as UML stereotypes in MATA and as domain-specific metadata for AoUCM and can be applied to any model element. In Fig. 2, the authentication aspect introduces the semantic marker <<*confidential*>>, the remote service aspect introduces the semantic markers <<*local*>> and <<*remote*>>, and the caching aspect introduces the semantic marker <<*cache*>>. Note that since semantic markers are also just another model element, they can also be used in aspect pattern matching.

Second, when an aspect is applied, the composition mechanism of MATA or AoUCM yields a new scenario model. In this composed scenario model, model elements may be tagged by several semantic markers. If more than one semantic marker is present on a model element, a conflict may be indicated. Therefore, each of these model elements is further investigated in the next step with the help of a goal model.

Third, a separate model defines the relationship between semantic markers in different domains. This model is used to identify semantic interactions between aspects. For example, there may be a relationship indicating that, in general, performance is negatively affected by security mechanisms. We represent this model as a goal model. A goal modeling language is used because there are well-defined relationships, as well as tool support, for specifying influences such as "hurts" (a negative impact) or "helps" (a positive impact).

Fourth, the elements in the goal model that represent semantic markers are instantiated based on the semantic markers of model elements in the composed model with more than one semantic marker. The goal model is then analyzed (evaluated) to see whether there is a potential conflict between such markers.

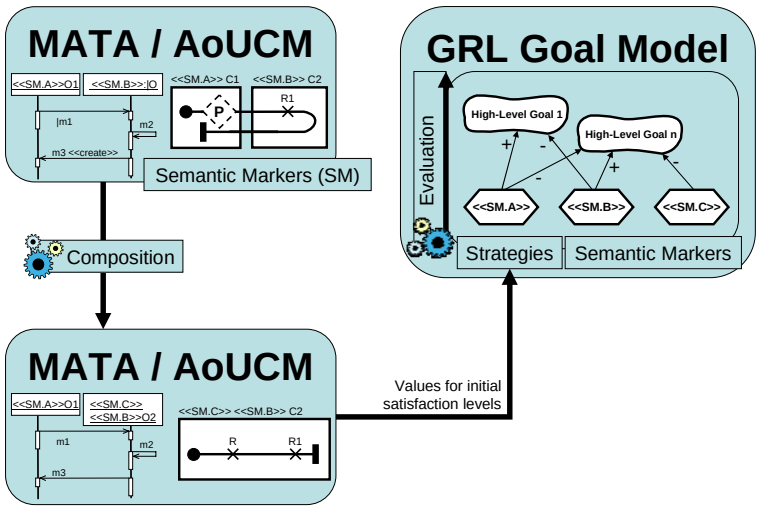

Fig. 3. Semantic-based aspect interaction detection.

In Fig. 2, the Authentication Server is <<*confidential*>> with only the Authentication aspect applied. By adding the Remote Service aspect, a <<*local*>> and a <<*remote*>> Authentication Server are introduced, both of which are still

<<*confidential*>>. Finally, if the Caching aspect is also applied, then the <<*confidential*>> <<*local*>> Authentication Server is additionally tagged with <<*cache*>>. The goal model can be analyzed to see if there is a conflict between <<*confidential*>>, <<*local*>> and/or <<*cache*>>. In this case, the user will be notified of a potential conflict between caching and confidentiality.

The process requires two additional activities to be undertaken by the modeler:

1. Each domain requires a set of domain-specific semantic markers that must be applied to aspect model elements.
2. There must be a goal model describing relationships between markers from different domains.

We expect ultimately that this additional modeling effort can be carried out once and then reused across many different applications. Aspects should be defined in as reusable a way as possible and then customized to a particular application context. Hence, we advocate the development of generic aspects each annotated with semantic markers. This means that the semantic marking need not be derived for each application. In a similar manner, the goal model need only be defined once and only needs to be updated incrementally when a new aspect is added. The result of the process described in Fig. 3 is a list of potential conflicts. Note that these are only *potential* conflicts because the semantic information provided by the markers is limited. It is up to the modeler to make a final decision on how to address the conflicts.

We make use of the Goal-oriented Requirement Language (GRL) to model and analyze the required goal model as explained in the next section.

4.2 Goal-oriented Requirement Language

We briefly present GRL and then illustrate the overall approach using the ongoing example. GRL combines the Non-Functional Requirements (NFR) framework [14] and i* framework [15] to support reasoning about goal models. The syntax of GRL (Fig. 4) is based on the syntax of the i* framework. A GRL goal graph is an AND/OR graph of *intentional elements* that optionally reside within an *actor boundary*. An *actor* represents a stakeholder of the system. A goal graph shows the high-level business goals and non-functional requirements of interest to a stakeholder and the alternatives for achieving these high-level elements. Often, alternatives represent functional properties of the system. A goal graph also documents *beliefs* (rationales) important to the stakeholder. Intentional elements can be softgoals, goals, tasks, and resources. *Softgoals* differentiate themselves from *goals* in that there is no objective measure of satisfaction for a softgoal. In general, softgoals are related more to NFRs, whereas goals are related more to functional requirements. *Tasks* represent solutions to (or *operationalizations* of) goals or softgoals. In order to be achieved, softgoals, goals, and tasks may require *resources* to be available.

Various kinds of *links* connect the elements in a goal graph. Decomposition links allow an element to be decomposed into sub-elements. AND as well as OR decompositions are supported. Contribution links indicate desired impacts of one element on another element. A contribution link has a qualitative contribution type (Fig. 4.d). Correlation links are similar to contribution links, but describe side effects

rather than desired impacts. Finally, dependency links model relationships between actors (one actor depending on another actor for something).

Our technique uses GRL goal models as the reasoning framework for aspect interaction detection. While the scenario models in MATA or AoUCM describe the software system, the GRL goal model describes the complex semantic dependencies between the aspects that are being considered for the software system. GRL supports reasoning about goals, as it shows the impact of often conflicting goals and various alternative solutions proposed to achieve the goals. The solutions in our case are the semantic markers that identify the aspect domains, while the high-level goals are the non-functional requirements associated with the aspects. A GRL *strategy* describes a particular configuration of alternative solutions in the GRL model and consists of a set of initial satisfaction ratings for elements in the GRL model. Typically, the satisfaction level (Fig. 4.c) of a chosen alternative is set to Satisficed. From the NFR framework, GRL borrows the notion of an *evaluation mechanism* [8] that propagates these low-level decisions regarding alternatives to satisfaction ratings of high-level goals, thus providing an assessment of the suitability of the proposed solution. Several strategies can be defined for a goal model, allowing trade-off analyses to be performed by exploring and comparing various configurations of alternatives. A more complete coverage of the notation elements is available in [9, 16].

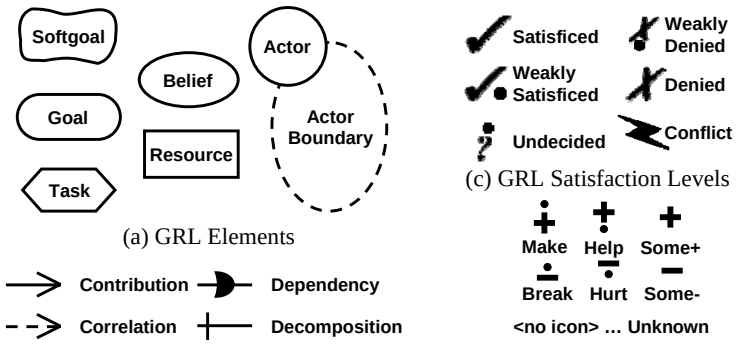

Fig. 4. Basic elements of GRL notation.

4.3 Electronic Voting Example

We now resume the presentation of the EVS example. The sequence diagram in Fig. 5 and use case map in Fig. 6 show the complete composed scenario model of the electronic voting machine introduced in Section 2 with all three aspects – Authentication, Remote Service, and Caching – applied, thus illustrating which semantic markers are merged onto which components. Note that an end point connected to a start point in a UCM acts like a sequence.

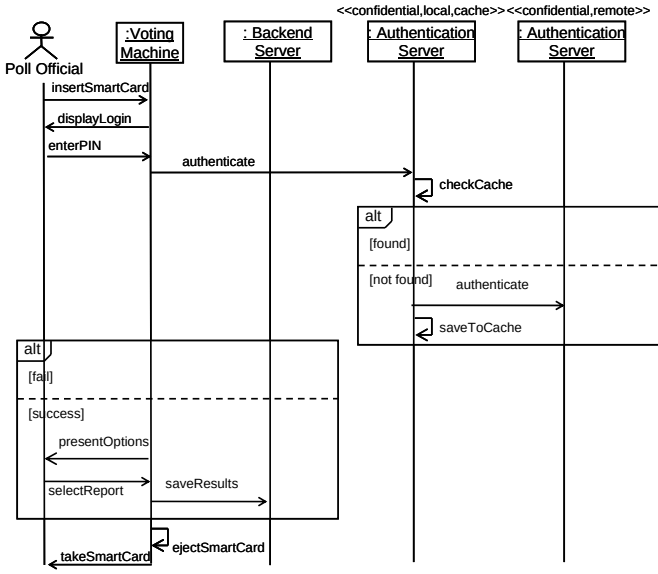

Fig. 5. Sequence diagram for composed electronic voting machine.

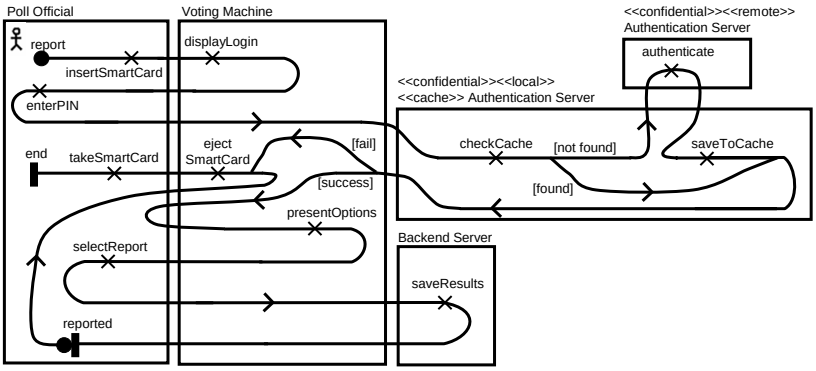

Fig. 6. Use case map for composed electronic voting machine.

The goal model for this set of domains is shown in Fig. 7. There are three high-level goals – Confidentiality, Consistency, and Performance – and five tasks that correspond to the semantic markers *<<confidential>>*, *<<remote>>*, *<<local>>*, *<<cache>>* and *<<encrypted>>*. The contribution links and contribution types in the goal model indicate that Authentication has a positive impact on Confidentiality. Remote Server as well as Caching, however, negatively impact Confidentiality, because data transferred across a network as well as cached data is potentially vulnerable to security attacks. Encryption, on the other hand, ensures that Confidentiality is achieved. Caching improves performance and therefore has a positive impact on Performance, but both Encryption and Remote Service in general result in performance penalties because of additional processing and network delays,

respectively. Authentication, on the other hand, does not have considerable positive or negative performance implications and therefore does not impact Performance. In terms of Consistency, Authentication as a non-remote service is problematic, because the distribution and update of data to local machines needs to be managed at setup time. A Remote Service with its tagged tasks Remote Server or Local Server, however, ensures that Consistency is achieved as the most-up-to-date information is always accessed.

The composed models in Fig. 5 and Fig. 6 show two model elements with more than one semantic marker – there are two authentication servers, one marked as confidential and remote, and one marked as confidential, local and caching. We call model elements with more than one marker *potential conflict* elements. The GRL model can now be used to analyze the implications of potential conflict elements.

For each potential conflict element, its marking is converted to a GRL strategy as follows. A task in the GRL model is given the maximum satisfaction value (100) if all its semantic markers are present on the potential conflict element. All other tasks are set to zero. After this, GRL's propagation algorithm is used to propagate satisfaction levels throughout the GRL graph.

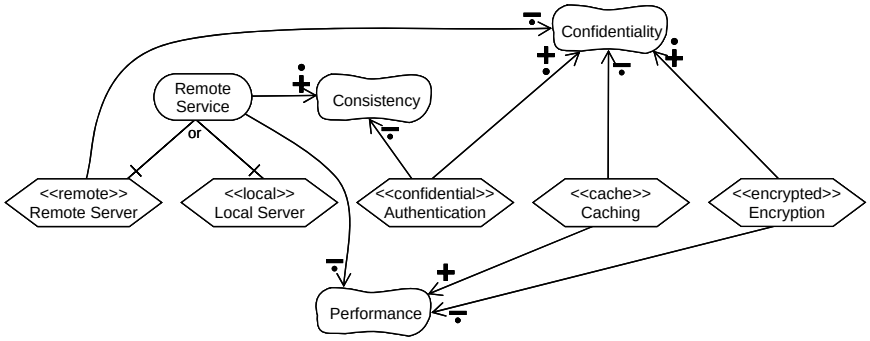

Fig. 7. Goal model for electronic voting machine.

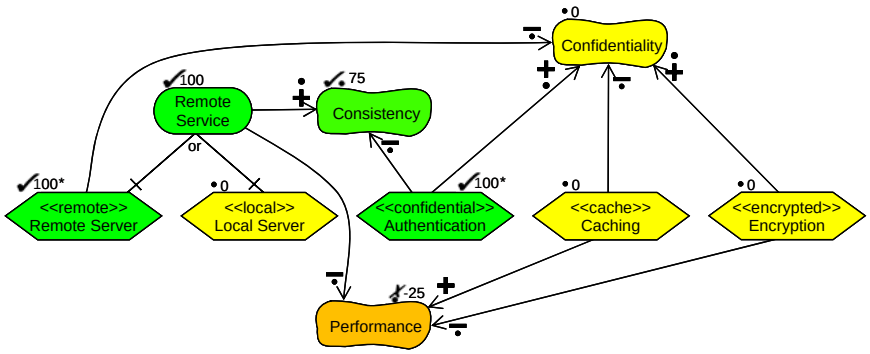

Fig. 8. <<*confidential*>> <<*remote*>> Authentication Server.

Fig. 8 shows the resulting GRL for the authentication server tagged with <<*confidential*>> and <<*remote*>>. Initial satisfaction levels are indicated by a *****

next to the satisfaction level. The markers for the Remote Server task and the Authentication task in the GRL are present, so these tasks are given an initial satisfaction level of 100. All other tasks are set to zero. The propagation algorithm (results shown in Fig. 8) determines that consistency is reasonably satisfied (value 75) whereas confidentiality is only partially satisfied (value 0), and performance may be problematic (value -25). Fig. 9 shows the results for the other authentication server.

The propagated results for the high-level goals of the two strategies are shown in column two and three of Table 1. At this point, there is no strategy that satisfies sufficiently Confidentiality and Performance at the same time. This is a sign to the modeler that there are aspect conflicts that must be addressed. The modeler could decide to add encryption to increase confidentiality at the cost of performance between the local and remote servers and also to the cache. If a new Encryption aspect is composed with the existing model, the local Authentication Server would have the markers: <<*confidential*>> <<*local*>> <<*cache*>> <<*encrypted*>>. The remote Authentication Server would have the markers: <<*confidential*>> <<*remote*>> <<*encrypted*>>. The propagated results are shown in Table 1.

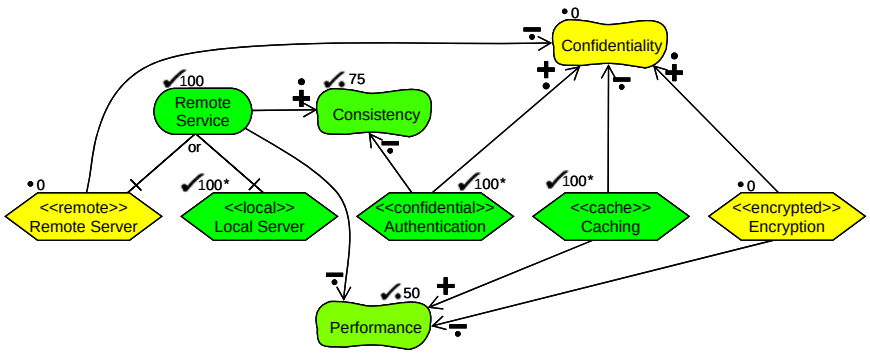

Fig. 9. <<*confidential*>> <<*local*>> <<*cache*>> Authentication Server.

Table 1. Propagated results for Authentication Server.

	confidential remote	*confidential local cache*	*confidential remote encrypted*	*confidential local cache encrypted*
Confidentiality	0	0	100	100
Consistency	75	75	75	75
Performance	-25	50	-50	25

5 Related Work

Despite a large body of work on aspect-oriented modeling (e.g., [17-20]), there have been few attempts to handle aspect interactions during modeling. One approach has been to supply notations for explicitly documenting interactions, such as aspect

interaction templates [2], precedences [2, 3], and aspect interaction charts [21]. MATA uses a numeric ordering scheme to capture precedence [7]. Since these approaches only document interactions, they could usefully be combined with the work in this paper, which instead detects interactions automatically.

As noted earlier, MATA employs critical pair analysis to detect syntactic interactions [7]. In [22], the authors use Alloy [23] to detect interactions between aspects written in the Aspect-UML language. The approach, however, requires formal pre- and post-condition specifications and thus is not as lightweight as our approach. At the programming level, such work is more mature. Typical approaches apply static analysis to detect shared joinpoints (e.g., [5, 24]). More recently, some work has tried to go beyond shared joinpoints to detect control flow-based interactions [25].

We are not aware of any work that takes into account the semantics of model elements when detecting interactions. The idea of semantically-informed aspect development, however, builds upon previous work in semantic-based aspect weaving. For example, in aspect-oriented requirements engineering, Chitchyan et al [26] use natural language processing to take into account English semantics when composing textual documents. For modeling, Klein et al [17] weave UML sequence diagrams by matching semantically equivalent but syntactically different sequences.

The aspect interaction problem is, of course, similar to the feature interaction problem [1]. Once again, however, we are unaware of the use of semantic annotations as in our approach. Rather, approaches are typically based on detecting structural interactions (e.g., [27]) or on applying formal methods, such as model checking [28].

6 Conclusion

This paper presented the first steps towards an approach for semantically detecting interactions between aspect models. The overall aim is to provide guidance to model developers as to how to build and compose aspect-oriented models. Tool support for the approach is provided by the MATA tool [11] for UML sequence diagrams and jUCMNav [29] for use case maps. Currently, not all parts of the process have been automated. jUCMNav includes a graphical editor for use case maps and for GRL goal models and support for semantic markers in jUCMNav is provided. Right now, however, the propagation algorithms in GRL do not take semantic markers into account and so the connection between goals in the GRL model and the semantic markers has to be done manually. It is straightforward to implement this in GRL.

Clearly, there is additional modeling effort required by our approach, and, in the future, empirical studies are needed to compare the benefits versus the additional effort required. The intention, however, is that aspect models will be defined in a reusable and generic manner so that, for a given application, the markup effort is minimal. In a similar way, the GRL models can be defined incrementally and reused in many different contexts. The use of domain-specific semantic markers is in keeping with a general trend in modeling to use domain-specific abstractions and, therefore, we are not suggesting a radical shift in the way that models are developed. Our technique does require, however, domain-specific annotation languages to be developed for each aspect domain. We would expect existing domain-specific

languages (e.g., UML profiles) to be usable directly. Hence, the intention is not to develop a new set of profiles but to use existing standardized profiles wherever possible.

References

[1] L. du Bousquet and J.-L. Richier, *Feature Interactions in Software and Communication Systems IX (ICFI)*: IOS Press, 2007.

[2] F. Sanen, N. Loughran, A. Rashid, A. Nedos, A. Jackson, S. Clarke, E. Truyen, and W. Joosen, "Classifying and Documenting Aspect Interactions," in *Workshop on Aspects, Components and Patterns for Infrastructure Software at AOSD*, Bonn, Germany, 2006.

[3] J. Zhang, T. Cottenier, A. Van den Berg, and J. Gray, "Aspect Composition in the Motorola Aspect-Oriented Modeling Weaver," *Journal of Object Technology*, vol. 6, pp. 89-108, 2007.

[4] P. Shaker and D. Peters, "Design-Level Detection of Interactions in Aspect-Oriented Systems," in *Aspects, Dependencies and Interactions Workshop at ECOOP 2006*, 2006.

[5] R. Douence, P. Fradet, and M. Südholt, "Composition, reuse and interaction analysis of stateful aspects," in *Aspect Oriented Software Development*, 2004, pp. 141-150.

[6] D. Balzarotti, A. Castaldo, and M. Monga, "Slicing AspectJ Woven Code," in *4th Workshop on Foundations of Aspect-Oriented Languages*, Chicago, USA, 2005.

[7] P. Jayaraman, J. Whittle, A. Elkhodary, and H. Gomaa, "Model Composition in Product Lines and Feature Interaction Detection using Critical Pair Analysis," in *International Conference on Model Driven Engineering, Languages and Systems (MODELS)*, Nashville, TN, 2007.

[8] ITU-T, "User Requirements Notation (URN), ITU-T Draft Recommendation Z.151", Geneva, Switzerland, September 4, 2008 (http://www.UseCaseMaps.org/urn) (accessed September 2008).

[9] D. Amyot, "Introduction to the User Requirements Notation: Learning by Example," *Computer Networks*, vol. 42, pp. 285-301, 2003.

[10] G. Mussbacher, D. Amyot, J. Whittle, and M. Weiss, "Flexible and Expressive Composition Rules with Aspect-Oriented Use Case Maps (AoUCM)," in *Early Aspects Workshop at AOSD 2007*, vol. LNCS 4765, pp. 19-38, 2007.

[11] J. Whittle and P. Jayaraman, "MATA: A Tool for Aspect-Oriented Modeling Based on Graph Transformation," in *Models in Software Engineering: Workshops and Symposia at MODELS 2007*, 2008, pp. 16-27.

[12] T. Kohno, A. Stubblefield, A. Rubin, and D. Wallach, "Analysis of an Electronic Voting System," in *IEEE Symposium on Security and Privacy*: IEEE Computer Society Press, 2004, pp. 27-40.

[13] G. Mussbacher, D. Amyot, and M. Weiss, "Visualizing Early Aspects with Use Case Maps," *Transactions on Aspect-Oriented Software Development III*, vol. LNCS 4620, pp. 105-143, 2007.

[14] L. Chung, B. A. Nixon, E. Yu, and J. Mylopoulos, *Non-Functional Requirements in Software Engineering*: Kluwer Academic Publishers, 2000.
[15] E. Yu, "Modeling Strategic Relationships for Process Reengineering," PhD thesis, Department of Computer Science, University of Toronto, Canada, 1995.
[16] J.-F. Roy, "Requirements Engineering with URN: Integrating Goals and Scenarios," MSc. thesis, OCICS, University of Ottawa, Canada, 2007.
[17] J. Klein, L. Hélouët, and J.-M. Jézéquel, "Semantic-Based Weaving of Scenarios," in *Aspect-Oriented Software Development (AOSD)*, Vancouver, Canada, 2006, pp. 27-38.
[18] R. France, I. Ray, G. Georg, and S. Ghosh, "Aspect-oriented approach to early design modeling," *IEE Proceedings - Software*, vol. 151, pp. 173-186, 2004.
[19] S. Clarke and E. Baniassad, *Aspect-Oriented Analysis and Design: The Theme Approach*: Addison Wesley, 2005.
[20] T. Cottenier, A. van den Berg, and T. Elrad, "Motorola WEAVR: Model Weaving in a Large Industrial Context," in *Aspect-Oriented Software Development (AOSD)*, Vancouver, Canada, 2007.
[21] S. Bakre and T. Elrad, "Scenario based resolution of aspect interactions with aspect interaction charts," in *10th International Workshop on Aspect Oriented Modeling (at AOSD)*, Vancouver, Canada, 2007, pp. 1-6.
[22] F. Mostefaoui and J. Vachon, "Design-level Detection of Interactions in Aspect-UML models using Alloy," *Journal of Object Technology*, vol. 6, pp. 137-165, 2007.
[23] D. Jackson, *Software Abstractions: Logic, Language and Analysis*: MIT Press, 2006.
[24] R. Douence, T. Fritz, N. Loriant, J.-M. Menaud, M. Segura-Devillechaise, and M. Sudholt, "An Expressive Aspect Language for System Applications with Arachne," in *Aspect-Oriented Software Development (AOSD)*, Chicago, Illinois, 2005, pp. 27-38.
[25] B. de Fraine, P. D. Quiroga, and V. Jonckers, "Management of Aspect Interactions using Statically Verified Control Flow Relations," in *Workshop on Aspects, Dependencies and Interactions (at ECOOP)*, 2008.
[26] R. Chitchyan, A. Rashid, P. Rayson, and R. Waters, "Semantics-Based Composition for Aspect-Oriented Requirements Engineering," in *Aspect-Oriented Software Development (AOSD)*, Vancouver, Canada, 2007, pp. 36-48.
[27] J. Liu, D. Batory, and S. Nedunuri, "Modeling interactions in feature oriented systems," in *International Conference on Feature Interactions (ICFI)*, 2005.
[28] L. du Bousquet, "Feature Interaction Detection using Testing and Model Checking: Experience Report," in *World Congress on Formal Methods in the Development of Computing Systems*, 1999, pp. 622-641.
[29] jUCMNav website. http://jucmnav.softwareengineering.ca./jucmnav (accessed June 2008).