

# Semantic-Based Interaction Detection in Aspect-Oriented Scenarios

Gunter Mussbacher  
SITE, University of Ottawa  
800 King Edward  
Ottawa, ON, K1N 6N5, Canada  
gunterm@site.uottawa.ca

Jon Whittle  
Dept. of Computing, InfoLab21,  
Lancaster University, Bailrigg,  
Lancaster, LA1 4YW, UK  
whittle@comp.lancs.ac.uk

Daniel Amyot  
SITE, University of Ottawa  
800 King Edward  
Ottawa, ON, K1N 6N5, Canada  
damyot@site.uottawa.ca

## Abstract

*Interactions between dependent or conflicting aspects are a well-known problem with aspect-oriented development (and related paradigms). These interactions are potentially dangerous and can lead to unexpected or incorrect results when aspects are composed. To date, most aspect interaction detection methods have been based either on purely syntactic comparisons or have relied on heavyweight formal methods. We present a new approach that is based instead on lightweight semantic annotations of aspects. Each aspect is annotated with domain-specific markers and a separate influence model describes how semantic markers from different domains influence each other. Automated analysis can then be used both to highlight semantic aspect conflicts and to trade-off aspects. We apply this technique to early aspects, namely, aspect scenarios, because it is desirable to detect aspect interactions as early in the software lifecycle as possible. We evaluate the technique using an industrial case study and show that the technique detects interactions that cannot be discovered using syntactic techniques.*

## 1. Introduction

A long-standing issue in software engineering is the *feature interaction problem* [1], i.e., how to identify and manage functionalities which interfere with each other. With the rise of aspect-oriented development (AOSD), this problem has found new relevance because aspects may interact in ways similar to features.

Aspects can be modeled at any stage of the software lifecycle [2]. We consider interactions between aspect-oriented scenarios. A scenario is an actual or expected execution trace of a system. Scenarios are a well-established technique in requirements engineering

because they are easily understood by a variety of stakeholders. Aspect-oriented scenarios are scenarios which crosscut other scenarios [3]. Scenarios can be represented in many ways – textually or by modeling languages. In this paper, we capture scenarios using UML sequence diagrams and hence place this work in the phase of requirements analysis. However, the work can potentially be applied to aspects in other phases and so we will use the term aspect-oriented modeling to refer to aspect models at any lifecycle stage.

In aspect-oriented modeling, interactions manifest themselves when multiple aspects affect a given point in the base model. In simple cases, aspects may be ordered so that they are applied in a way which respects their dependencies. In complex cases, there may be deep semantic conflicts between aspects that require a) a rethinking of which aspects should be applied or b) a remodeling of the aspects themselves. An example of the simple case is where an aspect assumes certain elements in the base that can only be introduced by another aspect. An example of the complex case is a conflict between two non-functional aspects where there are inherent trade-offs that arise irrespective of the ordering of aspects. A security aspect, for instance, will inevitably affect a performance aspect. Conversely, a performance aspect can affect a security aspect if the performance aspect caches results, which must then be protected. The simple case is a *syntactic interaction* because it can be detected by comparing syntax; the latter example is a *semantic interaction*.

The aspect interaction problem is still largely unsolved. One approach has been to explicitly document aspect interactions [4, 5]. This is useful but does not offer automated help in detecting interactions. Formal methods (e.g., model checking [6] or static analysis [7]) have been applied to detect interactions, but are effort-intensive. In previous work, we applied critical pair analysis to detect syntactic interactions between aspect models [8]. This works well for syntactic interactions but cannot handle semantic interactions.

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 tagged as `<<local>>` or `<<remote>>`. In a security aspect model, a server that stores sensitive data might be tagged 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 an issue because confidential data sent across a network must 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 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 aspects from different domains are applied. In this paper, we represent influences between semantic annotations as a goal model, written in the User Requirements Notation [9]. This allows us to leverage existing analysis techniques for goal models to reason about trade-offs between conflicting aspects.

The focus in this paper is on models used for scenario-based requirements analysis. We demonstrate the approach using aspect-oriented UML sequence diagrams [10].

The paper is organized as follows. Section 2 presents an example used throughout the paper as an illustration. Section 3 motivates the problem and Section 4 presents the new approach. Section 5 evaluates the approach on an industrial case study. Section 6 describes related work and conclusions are in Section 7.

A preliminary version of these ideas has been presented in workshop form [11]. This paper expands on the technique and presents an industrial validation.

## 2. Example: Electronic Voting

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

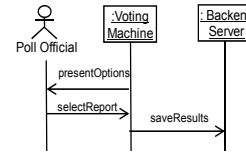


Fig. 1. Reporting use case

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 scenario captured as a standard UML sequence diagram (SD).

The Voting Machine in Fig. 1 is responsible for presenting reporting options. We will now consider how to apply three aspect scenarios to the base model:

- **Smart-card based authentication aspect**, which authenticates an actor 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 are good candidates for aspects because they tend to cut across multiple base scenarios. For example, although we only show one base model here, authentication is required in multiple base models when considering the entire system operation.

The three aspect scenarios will be modeled and then applied to the base scenario so that: (1) polling officials are authenticated; (2) authentication is done by querying a remote server; (3) authentication results are cached locally (since a polling official may need to access an EVS multiple times).

Fig. 2 shows aspect-oriented SDs for the three aspects. The notation used for SD aspects is that of the MATA aspect modeling tool [10]. In MATA, 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). 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.

MATA makes use of a simple UML profile consisting of stereotypes `<<create>>`, `<<delete>>`, and `<<context>>`. The *pattern* consists of any elements either 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>>`. If `<<create>>` is applied to a container element, such as an interaction fragment, it is by default applied to all

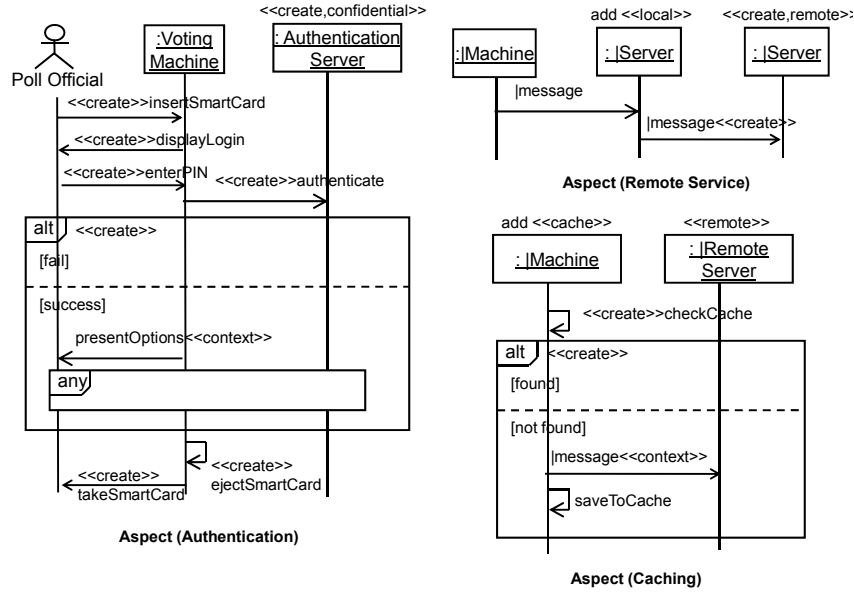


Fig. 2. Authentication, remote service, and caching aspects

contained elements unless a contained element is marked with `<<context>>`. When an aspect creates a stereotype, `add <<stereotype>>` is used instead of `<<create>>` because `<<create>>` is already a stereotype. Atomic variables in MATA are preceded by a vertical bar '|'. Sequence variables are given by a special interaction fragment with operator **any** which matches against any sequence of messages.

When applying the aspects in Fig. 2 to the base scenario in Fig. 1, suppose that the aspects are applied in the order: Authentication, Remote Service, Caching. Authentication matches against `presentOptions` and any messages following it in the base scenario (this is the *pattern*). Note that `presentOptions` is stereotyped as `<<context>>` to avoid it inheriting the `<<create>>` stereotype from the new `alt` fragment. The Authentication aspect adds messages around `presentOptions` and other messages existing in the base (as given by `<<create>>`) to introduce functionality to authenticate the poll official. When creating the Authentication Server, it marks it as `<<confidential>>`.

The Remote Service aspect matches against any message between any two objects (`|Machine` and `|Server`) and duplicates the message so that the copy is sent to a `<<remote>>` object. This is a rather general aspect and so must be instantiated to this particular context in MATA. This is done by assigning the value `authenticate` to the variable `|message` when applying the aspect. The effect of the aspect therefore is to make the existing authentication server local and have the local server forward the message to a remote server.

Finally, the Caching aspect matches any message sent to a `<<remote>>` object. The aspect converts the

sending object into a cache and adds messages for dealing with local caching.

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. 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. In previous work [8], we applied critical pair analysis (CPA), a graph transformation analysis technique, to detect syntactic interactions between MATA aspects. Syntactic interactions 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. They 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 these kinds of interactions.

As an example, consider the Remote Service and Caching aspects from Fig. 2. There is a clear dependency between these two aspects since Caching needs to match against an existing `<<remote>>` object and

Remote Service creates this object. That is, there needs to be a remote server before it makes sense to cache.

CPA is limited to detecting *syntactic* interactions because it is based solely on an analysis of the patterns used to define where 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 transmitted data 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 automatically flag such semantic conflict situations.

Intuitively, our technique works by composing the base with the aspects 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 *influence model* that describes relationships between these markers, the modeler can be notified of a possible problem: in this case, that sending confidential data to a remote server may require additional functionality, such as encryption.

#### 4. Semantic-Based Interaction Detection

Our technique relies on a set of domain-specific *semantic markers* for each aspect domain, as well as an *influence model* of how semantic markers from different domains influence each other. The overall approach is illustrated in Fig. 3. In step 1, a set of aspect scenarios is defined in MATA as well as base scenarios with which these aspects will be composed. Semantic markers (SM) are used to annotate each aspect scenario, thus adding semantic meaning. The semantic markers are implemented as UML stereotypes in MATA and can be applied to any model element.

In step 2 when aspects are applied, the composition mechanism of MATA (see [10] for full details) yields a new set of scenarios. In these composed scenarios, model elements may now be tagged by semantic markers from several domains. Step 3 takes as input a separate influence model that defines the relationships between semantic markers in different domains. For example, there may be a relationship indicating that, in general, performance is negatively affected by security mechanisms. We use a goal model for the influence model because there are well-defined relationships for

specifying negative or positive influences. Furthermore, reasoning about potential semantic interactions is made possible by automated tool support for evaluating a goal model based on these relationships.

Step 3 prepares the influence model for evaluation by instantiating it based on the existence of semantic markers in the composed scenario model and influence model. Finally, step 4 analyzes the influence model for potential conflicts between semantic markers.

In the voting machine example, during step 1, the Authentication aspect in Fig. 2 introduces the semantic marker `<<confidential>>`; Remote Service introduces `<<local>>` and `<<remote>>`; and Caching introduces `<<cache>>`. Note that since semantic markers are also just another model element, they can also be used in aspect pattern matching. In step 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>>`. After instantiating the influence model in step 3, it can be analyzed in step 4 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 the modeler to undertake two additional activities: 1) each domain requires a set of domain-specific semantic markers that must be applied to aspect model elements; 2) there must be an influence model describing relationships between markers from different domains. Given these, the composition, instantiation, and evaluation (steps 2-4) are automated.

We expect the additional modeling steps to be carried out once and then reused. Aspects should be defined in as reusable a way as possible and then customized to a particular application context (e.g., see [13]). 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 influence 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.

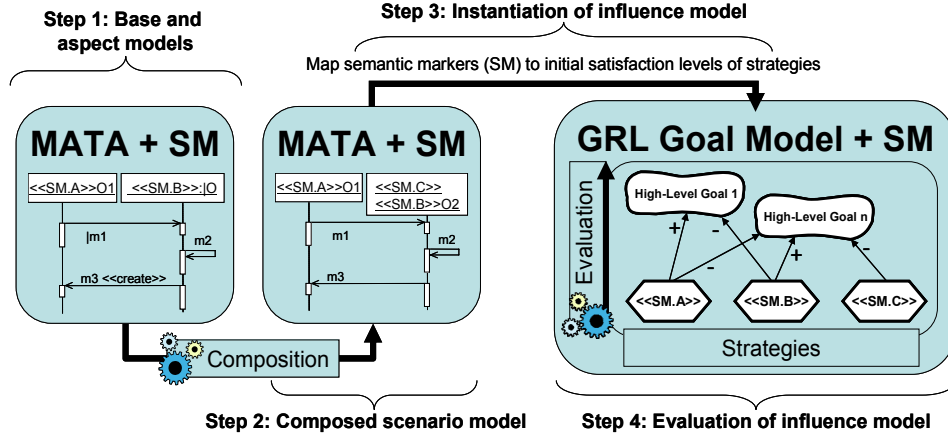


Fig. 3. Semantic-based aspect interaction detection

#### 4.1. Goal-oriented Requirement Language

The influence model (steps 3 and 4 in Fig. 3) is defined in our approach using goal modeling. In particular, we use the Goal-oriented Requirement Language (GRL) [9]. GRL combines the Non-Functional Requirements (NFR) framework and the  $i^*$  framework 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 a connected graph of *intentional elements* that 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. Softgoals, goals, and tasks are intentional elements used in our example. *Softgoals* differentiate themselves from *goals* in that there is no objective measure of satisfaction. 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.

Various kinds of *links* connect the elements in a goal graph. AND as well as OR-decomposition links allow an element to be decomposed into sub-elements. Contribution links indicate desired impacts of one element on another element. A contribution link has a

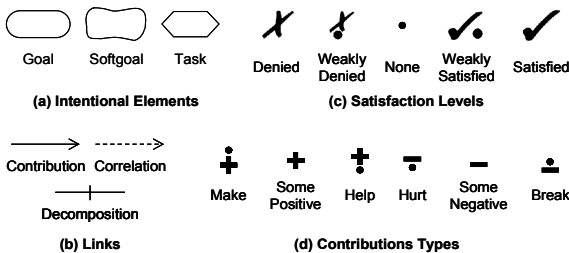


Fig. 4. Subset of GRL notation

qualitative contribution type (Fig. 4.d). Correlation links are similar to contribution links, but describe side effects rather than desired impacts.

Our technique uses GRL goal models as the reasoning framework for aspect interaction detection. 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 (i.e., tasks) in our case are each made up of one or more semantic markers, while the high-level goals are 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 quantitative and qualitative satisfaction values for GRL model elements. Typically, the satisfaction value (Fig. 4.c) of a chosen alternative is set to the maximum quantitative value 100 (and its corresponding highest qualitative value Satisfied). From the NFR framework, GRL borrows the notion of an *evaluation mechanism* [9] that propagates these

Table 1. Mapping for Influence Model

Concept	Influence Model
aspect	goal graph
NFR of aspect	softgoal
semantic marker	task
impact of semantic marker on its own aspect	contribution
impact of semantic marker on other aspect	correlation
potential conflict element (PCE)	strategy
all semantic markers of task A are present on PCE E	A's satisfaction value in E's strategy is set to 100
all semantic markers of task A are <i>not</i> present on PCE E	A's satisfaction value in E's strategy is set to 0

Note: This is generally a one-to-one mapping.

low-level decisions regarding alternatives to satisfaction values 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.

#### 4.2. Instantiating the Influence Model

The mapping from composed scenario models to the influence model is summarized in Table 1, as are general rules on how to create the influence model. For each aspect, a GRL goal graph is created that models a) the non-functional requirements (NFRs) associated with the aspect as *softgoals*, b) each aspect-specific semantic marker as a *task*, c) the impact of the markers on the aspect's NFRs as *contributions*, and d) the impact of the markers on other aspect's NFRs as *correlations*. If necessary, several markers may be combined into one task or intermediate *goals* may be added to group markers. For small systems, the separate aspect-specific goal graphs may be combined into one single goal graph.

Before the influence model can be evaluated, it must be instantiated. We call model elements with more than one marker *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 (Satisfied) if all its semantic markers are present on the potential conflict element. All other tasks are set by default to 0 (None). The GRL evaluation mechanism can now analyze the influence model by propagating satisfaction values throughout the model.

#### 4.3. Electronic Voting Example

The SD in Fig. 5 shows the complete composed scenario model of the EVS introduced in Section 2 with all three aspects – Authentication, Remote Service, and Caching – applied, thus illustrating which semantic markers are merged onto which components.

Two model elements with more than one semantic marker exist. Therefore, two GRL strategies will be

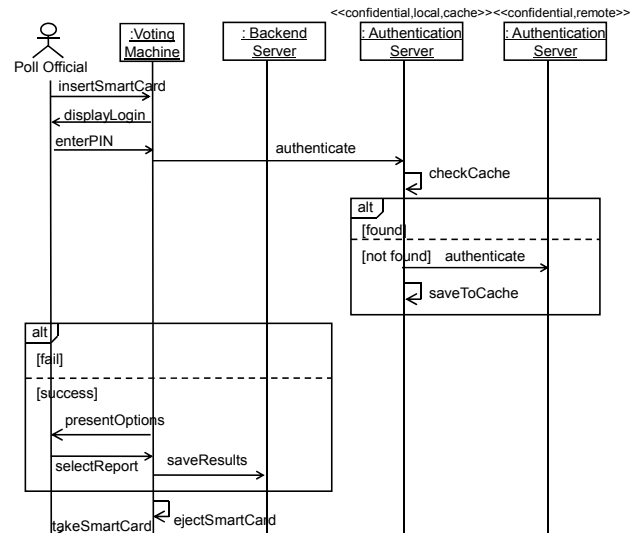
**Table 2. Propagated results for Authentication Server**

	<i>con rem</i>	<i>con local cache</i>	<i>con rem enc</i>	<i>con local cache enc</i>
Confidentiality	0	0	100	100
Consistency	75	75	75	75
Performance	-25	50	-50	25

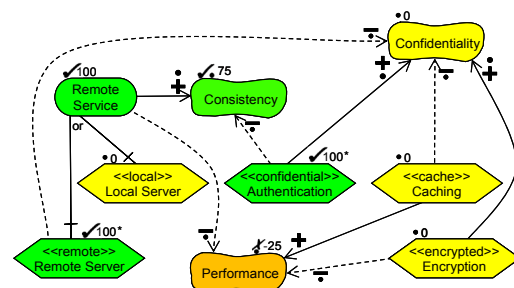
con ... confidential, enc ... encrypted, rem ... remote

created. Strategy A for the authentication server marked as confidential and remote, and strategy B for the authentication server marked as confidential, local, and caching. Fig. 6 shows the evaluation of the strategy for the authentication server tagged with `<<confidential>>` and `<<remote>>` with the influence model for the three domains listed above as well as Encryption.

Fig. 6 shows one single influence model for all four domains. The Authentication aspect is modeled by the Confidentiality softgoal and the task with the `<<confidential>>` tag; the Remote Service aspect by the Consistency softgoal and the tasks with `<<remote>>` or `<<local>>` tags; the Caching aspect by the Performance softgoal and the task with the `<<cache>>` tag; and finally the Encryption aspect by the Confidentiality softgoal and the task with the `<<encrypted>>` tag. The contribution/correlation links and contribution types in the influence model indicate that Authentication has a positive impact on Confidentiality (contribution type: Help). Remote Server as well as Caching, however, negatively impact Confidentiality (Hurt), because data transferred across a network as well as cached data is potentially vulnerable to security



**Fig. 5. SD for composed model**



**Fig. 6. Evaluated influence model**

attacks. Encryption, on the other hand, ensures Confidentiality is achieved (Make). Caching improves performance (SomePositive) and Authentication does not have considerable negative or positive performance implications and is therefore neutral (no link), but both Encryption and Remote Service in general result in significant performance penalties (Hurt) because of additional processing and network delays, respectively. In terms of Consistency, Authentication as a non-remote service is problematic (Hurt), 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 and Local Server, however, ensures that Consistency is achieved as the most-up-to-date information is always accessed (Make).

A \* next to the satisfaction value indicates initial satisfaction values. The markers for the Remote Server task and the Authentication task are present on the authentication server in the composed scenario model, so these tasks are given an initial value of 100 (Satisfied). All other tasks are set to 0 (None). The evaluation mechanism then determines all other values in the influence model, i.e., consistency is reasonably satisfied (value 75) whereas confidentiality is only partially satisfied (0), and performance may be problematic (-25).

The propagated results for the high-level goals of the strategies A and B are shown in the first two columns of Table 2. 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 the last two columns of Table 2.

## 5. Validation

To validate our detection technique, we applied it to an industrial case study, namely, a set of use cases and sequence diagram models describing a software defined radio application (for reasons of confidentiality, we cannot identify the source). This application is described by a 40-page document, which captures eleven primary use cases (as well as a number of auxiliary use cases) and eight scenarios for these use cases, giv-

en as UML sequence diagrams. The focus of the validation was to see if our technique was able to detect semantic interactions in practice and, furthermore, to see if it detected interactions that could not be found simply by applying syntactic interaction detection.

### 5.1. Methodology

The original document had not been developed using aspect-oriented modeling. Neither did it use semantic markers. Therefore, the first step was to refactor the sequence diagrams into aspects tagged with semantic markers. The steps of our validation experiment are summarized below:

1. Refactor original sequence diagrams to modularize crosscutting concerns. (This step proved interesting in itself because we easily identified a number of concerns that would benefit from modularization.)
2. Develop a set of semantic markers for each aspect domain and annotate the aspects with these markers.
3. Develop an influence model for these markers and related non-functional requirements.
4. Apply our techniques to detect syntactic interactions.
5. Apply our techniques to detect semantic interactions.

In step 2, we originally intended to use existing domain descriptions, e.g., UML profiles, as semantic markers. However, existing UML profiles proved either too complicated for our initial validation experiment, or did not cover the required domains. Note that we looked both for syntactic and semantic interactions in steps 4 and 5 so that we could compare the results.

After collecting the results from steps 4 and 5, we decided to classify the interactions to better understand them. For syntactic interactions, we identified *benign* and *non-benign* interactions. Benign interactions are those where the technique highlights an interaction between at least two aspects but no action is needed from the modeler to resolve it. This typically happens when the order of application of two aspects returns different results but where the modeler does not care about the order. For example, it is irrelevant which of two logging messages is carried out first. For semantic interactions, we classified the interactions according to the domains involved. A Type I interaction is where an aspect negatively impacts the performance of the overall application (a performance conflict). A Type II interaction is where an aspect stores data (e.g., in an audit database) that must be protected when a security aspect is applied (a storage/security conflict). A Type III interaction is where an aspect introduces new functionality

which implies changes to a security aspect such as the introduction of fault handling procedures that must be protected lest they become an easy target for attackers (a functionality/security conflict).

## 5.2. Results

After step 1 (refactoring), we obtained five sequence diagrams making up the base model and fourteen MATA sequence diagram aspects covering four aspect domains – see Table 3.

Fig. 7 summarizes the syntactic and semantic interactions detected. The set of syntactic and semantic interactions is disjoint, showing that semantic interaction detection highlights potential problems not found by syntactic detection alone. We could not classify the semantic interactions into benign and non-benign because this would require an application-specific analysis from the original model developers which was not available to us.

The results show that our techniques do indeed discover interactions not discoverable by our earlier efforts on syntactic interactions. The largest category of semantic interactions relate to the impact on performance (Type I). This is to be expected as almost any aspect is likely to affect performance. Type I interactions are therefore not as interesting as the other categories. It is in the Type II-III and Other categories that real interest lies. These highlight areas to the developer where there are trade-offs to be made. For example, an interaction was discovered between fault tolerance and authentication. This occurred because authentication introduces new considerations for fault tolerance that were not modeled by the fault tolerance application. A developer might need to add fault tolerance to the authentication server, for instance. Upon seeing this interaction, a developer would therefore be expected either

to decide that no fix is necessary (a benign interaction) or to add new modeling elements to capture fault tolerance for authentication (a non-benign interaction). Note that even detecting benign interactions is useful and necessary because the developer can at least be sure that s/he has not missed certain key considerations. The principal kinds of semantic interactions (other than Type I) that we detected were as follows: interactions between logging and security; interactions between security and fault tolerance; interactions between different security mechanisms – such as storing privilege rules which then must be protected; interactions between caching and security.

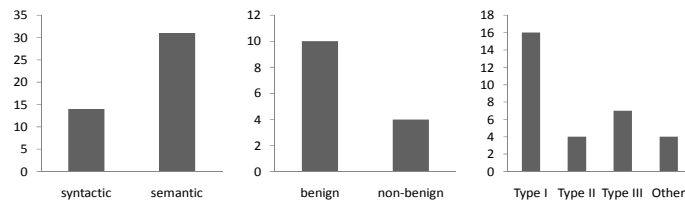
Note that the quality of the results depends on the semantic markers used to annotate the aspects. There is a choice in granularity level. For example, a <<security-critical>> annotation would return interactions with little information about where the trade-off lies. A better annotation set would split <<security-critical>> into <<access-control-by-password>>, <<authentication>>, <<non-repudiation>>, <<checksum>>, <<encryption>>, etc. In this experiment, we made use of a total of eleven markers: five for security, three for auditing, one for caching, and two for fault-tolerance. This allowed our technique to detect interactions within a single domain.

## 5.3. Lessons Learned

The case study presented here constitutes only a first step in a longer term planned validation effort. In terms of threats to internal validity, although the original models were developed independently by domain experts, we were responsible for refactoring into aspects and coming up with the influence model. We also derived a set of semantic markers ourselves instead of using available UML profiles which would significantly increase the number of stereotypes. In terms of external validity, this application is a security-critical system and may not generalize to information systems, for example. In particular, the models were detailed and well-specified. This might not be the case for information systems. However, despite the limitations, this initial experiment resulted in a number of lessons learned.

**Table 3. Aspects Summary**

id	Domain	Aspect
1A	Audit Trail	General Logging
1B	Audit Trail	Non-repudiation
2A	Security	Tamper Proof
2B	Security	Authorization
2C	Security	Authentication of Downloads
2D	Security	Decryption
2E	Security	Filtering
2F	Security	Authentication - General
2G	Security	Authentication of Keys
2H	Security	Encryption
3	Cache	Download
4A	Fault Tolerance	Reporting – Fault
4B	Fault Tolerance	Reporting – Failure
4C	Fault Tolerance	Retry



**Fig. 7. Summary of Results**

**Usefulness beyond interactions.** It turned out that semantic mark-up served a number of purposes other than simply detecting interactions. Two examples arose during the experiment. The first is where a single component receives many semantic markers, which suggests that perhaps its responsibilities should be refactored to improve modularity and reduce complexity. The second is where markers (and their combinations) could be used to auto-suggest design or analysis patterns. More generally, we feel there is scope for a range of uses of these semantic markers.

**Application during iterative modeling.** In this experiment, we examined only the final version of the models. However, we expect the technique to be even more useful during iterative development since the aim is really to assist modelers in developing good models.

**False positives.** Our technique is principally a heuristic one since the granularity of markers is rarely sufficient to definitively say that there is a conflict. For example, it can be the case that the tagging suggests a problem which never manifests itself in the models. This may happen if the models are written at a finer level of detail than the influence model.

**Granularity of markers.** As noted previously, the results will vary depending on the marker granularity.

**Return on investment.** It remains to be seen whether the additional effort in applying our technique – in terms of developing markers, influence models and interpreting the results – is worth the effort in practice. Our intention is that the markers and influence models can be reused across applications but the feasibility of this must still be demonstrated.

**Scalability and Complexity.** Scalability is only a modest problem, because the tags are applied to each aspect individually and not the composed model. Furthermore, the complexity of the influence model is directly related to the inherent complexity of the semantic interactions between aspects. The analysis time for the influence model, however, is generally negligible and certainly for the case study's influence model with eleven tasks, a few NFRs, and the links between them.

## 6. Related Work

Despite a large body of work on aspect-oriented modeling (e.g., [14-17]), 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 [4], precedences [4, 5], and aspect interaction charts [18]. MATA uses a numeric ordering

scheme to capture precedence [8]. 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 [8]. In [19], the authors use a formal language 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., [7, 20]). More recently, some work has tried to go beyond shared joinpoints to detect control flow-based interactions [21].

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. [22] use natural language processing to take into account English semantics when composing textual documents. For modeling, Klein et al. [14] weave UML SDs by matching semantically equivalent but syntactically different sequences. In the context of aspect-oriented programming, Bergmans [23] discusses the use of semantic annotations for composition filters.

For the feature interaction problem [1], we are unaware of the use of lightweight semantic annotations. Rather, approaches are typically based on detecting structural interactions (e.g., [24]) or on applying formal methods, such as model checking [25].

## 7. Conclusion

This paper presented an approach for semantically detecting interactions between aspect-oriented scenarios. 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 [10] and jUCMNav [26]. jUCMNav includes a graphical editor for GRL goal models, evaluation mechanisms, and support for semantic markers. MATA and jUCMNav are not currently fully integrated, but do allow proof-of-concept experiments to be undertaken. Furthermore, we have also applied these techniques successfully to another scenario-based notation (Use Case Maps [9]), which is supported in jUCMNav and for which aspect-oriented extensions have been recently developed [27].

Clearly, there is additional modeling effort required by our approach, and, in the future, more empir-

ical 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 influence 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, however, require domain-specific annotation languages to be developed for each aspect domain. We would hope that existing domain-specific languages (e.g., UML profiles) could be used directly. Hence, the intention is not to develop a new set of profiles but to use existing standardized profiles wherever possible.

**Acknowledgements.** Supported by NSERC Canada (Discovery Grants and Postgraduate Scholarships).

### References

- [1] M. Calder, M. Kolberg, E. H. Magill, and S. Reiff-Marganiec, "Feature interaction: a critical review and considered forecast", *Computer Networks*, 41, pp. 115-141, 2003.
- [2] A. Rashid, P. Sawyer, A. Moreira, and J. Araújo, "Early Aspects: A Model for Aspect-Oriented Requirements Engineering", in *10th Int. Conf. on Requirements Engineering*, 2002, pp. 199-202.
- [3] J. Araújo, J. Whittle, and D.-K. Kim, "Modeling and Composing Scenario-Based Requirements with Aspects", in *12th Int. Conf. on Requirements Eng.*, 2004, pp. 58-67.
- [4] F. Sanen, et al., "Classifying and Documenting Aspect Interactions", in *Workshop on Aspects, Components and Patterns for Infrastructure Software at AOSD*, Bonn, 2006.
- [5] 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, no 7, pp. 89-108, 2007.
- [6] P. Shaker and D. Peters, "Design-Level Detection of Interactions in Aspect-Oriented Systems", in *Aspects, Dependencies and Interactions Workshop at ECOOP*, 2006.
- [7] R. Douence, P. Fradet, and M. Südholt, "Composition, reuse and interaction analysis of stateful aspects", in *Aspect Oriented Software Development*, 2004, pp. 141-150.
- [8] P. Jayaraman, J. Whittle, A. Elkhodary, and H. Gooma, "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, pp. 151-164.
- [9] ITU-T, "Recommendation Z.151 (11/08): User Requirements Notation (URN) - Language Definition", Geneva, Switzerland, 2008.
- [10] 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*, H. Giese, Ed.: Springer, LNCS 5002, 2008, pp. 16-27.
- [11] G. Mussbacher, J. Whittle, and D. Amyot, "Towards Semantic-Based Aspect Interaction Detection", in *Workshop on Non-functional System Properties in Domain-Specific Modeling (at MODELS 2008)*, 2008.
- [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] J. Klein and J. Kienzle, "Reusable Aspect Models", in *11th Workshop on Aspect-Oriented Modeling (at MODELS 2007)*, 2007.
- [14] J. Klein, L. Hélouët, and J.-M. Jézéquel, "Semantic-Based Weaving of Scenarios", in *Aspect-Oriented Software Development (AOSD)*, Bonn, Germany, 2006, pp. 27-38.
- [15] 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.
- [16] S. Clarke and E. Baniassad, *Aspect-Oriented Analysis and Design: The Theme Approach*: Addison Wesley, 2005.
- [17] 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.
- [18] 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.
- [19] 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.
- [20] R. Douence, et al., "An Expressive Aspect Language for System Applications with Arachne", in *Aspect-Oriented Software Development (AOSD)*, Chicago, 2005, pp. 27-38.
- [21] B. de Fraigne, 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.
- [22] 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.
- [23] L. M. J. Bergmans, "Towards Detection of Semantic Conflicts between Crosscutting Concerns", in *AAOS Workshop at ECOOP 2003*, Darmstadt, Germany, 2003.
- [24] J. Liu, D. Batory, and S. Nedunuri, "Modeling interactions in feature oriented systems", in *International Conference on Feature Interactions (ICFI)*, pp. 178-197, 2005.
- [25] 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.
- [26] jUCMNav 3.2, <http://jucmnav.softwareengineering.ca/jucmnav> (accessed February 2009).
- [27] G. Mussbacher, "Aspect-Oriented User Requirements Notation", in *Models in Software Engineering: Workshops and Symposia at MODELS 2007*, H. Giese, Ed.: Springer, LNCS 5002, 2008, pp. 305-316.