Incorporating Security Behaviour into Business Models using a Model Driven Approach

Peter F. Linington and Pulitha Liyanagama
University of Kent Computing Laboratory
Canterbury, Kent, UK
{pfl,pll4}@kent.ac.uk

Abstract

There has, in recent years, been growing interest in Model Driven Engineering (MDE), in which models are the primary design artifacts and transformations are applied to these models to generate refinements leading to usable implementations over specific platforms. There is also interest in factoring out a number of non-functional aspects, such as security, to provide reusable solutions applicable to a number of different applications.

This paper brings these two approaches together, investigating, in particular, the way behaviour from the different sources can be combined and integrated into a single design model. Doing so involves transformations that weave together the constraints from the various aspects and are, as a result, more complex to specify than the linear pipelines of transformations used in most MDE work to date. The approach taken here involves using an aspect model as a template for refining particular patterns in the business model, and the transformations are expressed as graph rewriting rules for both static and behaviour elements of the models.

1 Introduction

In Model Driven Engineering (MDE), models are the primary design artifacts and transformations are applied to these models to generate refinements leading to usable implementations over specific platforms. The main emphasis in demonstrating the MDE concepts has been on the refinement of general designs to platforms as one of the steps in a code generation process.

At the same time, there has been considerable interest in factoring out from application designs a number of non-functional aspects, such as security, so that it becomes possible to provide and manage reusable solutions to them. These solutions can then be woven together with outline designs of the business logic for particular application requirements to produce solutions with the desired properties. The aim should be to maintain the separation of concerns, minimizing the need for additional labeling of the basic business model with aspect-specific markers.

This paper uses security as a specific case study, but the main aim is to use this example to explain the requirements placed on transformation techniques by support of the weaving process.

This paper brings these two approaches together, investigating, in particular, the way behaviour from the different sources can be combined and integrated into a single design model. This involves transformations to perform the weaving together of the constraints from the various aspects, and these transformations are, as a result, more complex to specify than the linear pipelines of transformations generally used to date. The approach taken here involves using an aspect model as a template for refining particular patterns in the business model, and the transformations are expressed as rewriting rules for both static and behaviour elements of the models.

The requirements for MDE tools to be used for the integration of security aspects have been reviewed in [14]. An initial case study carried out within the InterOp Network of Excellence was published in [12].

The remainder of this paper is organized as follows. In section 2 we give some background information on previous work on both model driven engineering and aspect oriented development, and in section 3 we set out a number of requirements and simple supporting mechanisms for managing security. In section 4 we introduce the style of transformation applied to the weaving of the business logic and security aspects, and in section 5 outline the implementation environment in use. Sections 6 and 7 then describe the detailed approach to structural and behavioural transformations respectively. Section 8 reviews the process described with regard to the initial objectives. Finally, section 9 draws conclusions and indicates some future directions for the work.

2 Previous work

2.1 Model driven techniques

The current model driven movement is the natural next stage of a steady trend towards stronger tool integration and progressively higher level representations of designs. This same process can be seen in the viewpoint separation of ODP [5] [6], and was espoused by the OMG in their millennial white paper on Model Driven Architecture [19]. OMG experts provided a specific framework for their MDA approach [16], in which they introduced the concepts of CIM, PIM and PSM to capture the trajectory from organizational to platform specific designs (see figure 1). Subsequently, via the QVT RFP, a transformation language for use within MDA has been formulated [17].

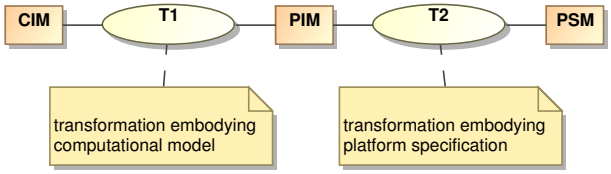

Figure 1. Standard MDA transformation sequence.

There are many current activity in the development and use of model driven tools, for example Atlas [1], Mola [8] and Tefkat [13]. The web site operated by the PlanetMDE organization provides an excellent summary of the broader spread of model drive engineering activities.

Earlier work attempted to perform the necessary transformations in a completely generic way using graph rewriting techniques and graph grammars. A powerful example is $ATоM^3$ [2]. The MDA tools differ from this in that they specialize the transformation to allow more domain-specific tailoring. The current proposal follows this direction of adding constraints to the transformation, but assumed that there are two kinds of modeller: those concerned with ensuring that broad domain specific rules are captured (these are incorporated in the transformation) and those concerned with specific designs, who should work in a uniform notation as close as possible to familiar UML usage. The UML fragments for specific designs are then linked by the transformations formulated by the specialists.

One of the main issues in the design of model transformation languages is the role of iteration and reverse engineering in the specification process. Directed, imperative languages give an easier implementation path, but declarative languages are potentially more concise and closer to the minimum required semantics. They also fit more naturally into the support for dynamically evolving system requirements. This tension is discussed further below when describing the details of the approach taken.

2.2 Aspect based techniques

The main thrust of this paper is the incorporation of security into application logic, but it raises issues that apply to a broader range of non-functional aspects. As such, it brings together a number of techniques linking model driven and aspect oriented approaches. Aspect orientation has been an active research thread for many years, dating originally from the work at Xerox Parc in the 1990s [10] [20], [11] which lead to the development of AspectJ [9].

The basic idea of an aspect is that in any set of applications there are a number of cross-cutting issues that represent recurring themes in the various designs. If these can be extracted and made the subject of a separately maintained fragment of specification, then they can be applied to all applications, both existing and newly designed, and so current best practice can be shared in an effective manner. We are interested particularly in the extraction of the non-functional aspects, which are only weakly linked to the detail of the business logic, such as performance, quality of service, business value, and, of course, trust and security.

The basic pattern in aspect oriented work is that in a base specification there are a number of potential join points at which additional behaviour can be integrated, from which an active set is extracted by application of a set of constraints known as a Pointcut. The weaving process then applies additional behaviour known as the aspect advice at the selected points.

Some work has already examined the integration of aspects and model driven techniques, such at [18]. However, the approach taken there has concentrated on the model driven preparation of primary and aspect specification followed by a standard aspect oriented software development weaving process. Here we integrate both steps within one transformational framework.

2.3 Modelling security

For completeness, it should be noted that there have been a number of proposals for enhancing general purpose modelling to support security aspects. Well known examples are UMLSec [7] and SecureUML [15], which is based on [4]. These are, in essence, UML profiles, and so weave specific security role information with the business logic in a way that this paper is trying to avoid.

Our aim is the separation of concerns, not weaving them together in a single extended language.

3 Target security aspects considered

Before we can concentrate on how, in detail, security is to be incorporated into business models, we need to identify where in a system configuration security problems occur. In general, this will involve the analysis of trust and threat models with particular emphasis on points at which information flows between domains with different security properties (see figure 2). This allows the key pieces of behaviour where security precautions are needed to be selected.

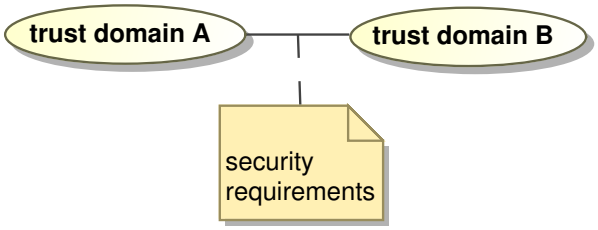

Figure 2. Trust analysis allows security requirements associated with domain boundaries to be identified.

If the key communication paths are known, then applying a suitable mechanism reduces to performing some transformation local to the selected paths and to the introduction of any associated shared infrastructure components, such as credential or policy repositories or domain specific decision points. As working examples in this paper, we consider

- the provision of secrecy by encryption of the transport supporting the information flow; the main requirement for the provision of credentials and marking of the flow as needing special treatment in a platform-specific way.
- provision of authentication on each interaction by introducing a gateway to control the information flow, based on additional parameterization, such as simply by provision of a user identity and password on each request(see figure 3).
- more efficient mechanisms in which an initial exchange validates credentials and returns some short-term token that can be included with each interaction making up an extended session (see figure 4); this requires analysis of the behaviour in the business logic so as to identify effective candidate sessions and to ensure that clients maintain some associated local token storage (see section 7.2).

Once the right trigger points have been identified and the appropriate mechanisms incorporated, care must be taken

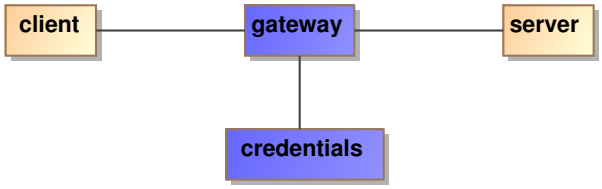

Figure 3. Providing password-based authentication on each operation.

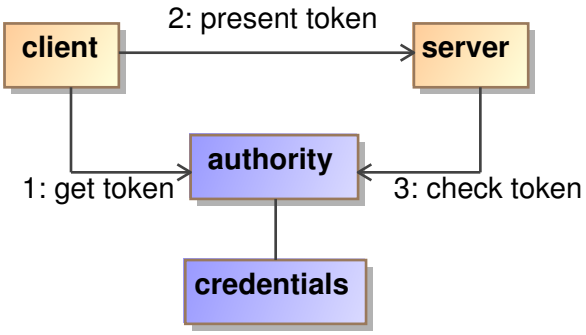

Figure 4. Providing session-token-based authentication on each operation.

that the target security properties are maintained. Non-functional aspects such as security and performance are particularly fragile, since further unrelated transformations can easily invalidate the desired properties by opening backdoors or invalidating assumptions about worst-case performance.

To overcome this problem, a transformation that establishes some required property should also at the same time generate constraints that verify the assumptions on which these properties are based. Thus, for example, if a performance requirement is met under the assumption that some processing element meets a particular real-time deadline, a constraint should be generated associated with that element so that there is an explicit check to ensure further transformations to meet other requirements do not invalidate it.

4 Approach to transformation

The traditional model driven engineering approach is based on the specification of a transformation that relates source and target models. The transformation is expressed in terms of the source and target metamodels, making it powerful and reusable, but at the cost of requiring considerable skill on the part of the transformation author.

To support the composition of business logic and non-functional aspects, a different form of transformation is needed. It would be possible, in principle, to codify each non-functional aspect as a specific transformation, but this would be hard for the author to do and difficult for others to understand. It would be much more accessible to capture the generic weaving process as a transformation that relates multiple models, and to represent the pointcuts and aspect advice as separate, more conventional, models. That is the approach taken here.

The business logic, the form of the pointcut and the advice are all separate models, with specific roles in a general weaving process; the aspect models steer the weaving process, and are more like patterns or templates than conventional design models. The transformation rules capture the specific details of the weaving process, which are, in general, dependent on the source and target metamodels. This involves the expression of complex constraints, particularly when considering the weaving of behaviour, where consequential changes may propagate a long way.

One of the issues to be considered in applying model driven engineering ideas is the trade-off between imperative and declarative modelling styles, associated with the balance between forward and reverse engineering processes. This becomes more complex in an aspect-oriented setting, because of the many-to-many nature of the transformation; there are more possible mutability markings, and more varieties of constraints in finding solutions consistent with them. We return to this issue below.

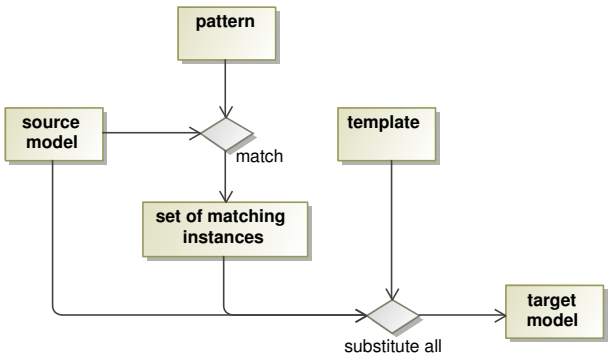

Figure 5. The model-merging process and the models involved in it.

The approach taken here is based on graph grammars [3], and involves two steps, illustrated in figure 5. The first is a matching process, in which the pattern model is compared to the source model; the matching process returns the set of all the situations in which the pattern corresponds to some part of the source model. The resulting set of matching instances is expressed as a set of labelings of the source model with role names in the match from the pattern model. A matching instance is effectively a view of the source model in which all unlabeled items are hidden, and which then satisfies the constraints from the pattern model on the elements that remain visible.

The second step constructs the target model by taking each matching instance in turn and using an instance of the template model to populate the corresponding fragment of the target model. This is done by substituting details from the matching instance for each occurrence of the pattern role names found in the template model. Finally, any source model elements not yet used in the substitution process are included by applying a default copying behaviour.

These two processes of matching and substitution are generic and are made up of common constraints expressed in terms of the metamodels of the source, pattern, template and target. Once the processes have been defined, they can be performed for any specific sets of these models that assume the same weaving semantics.

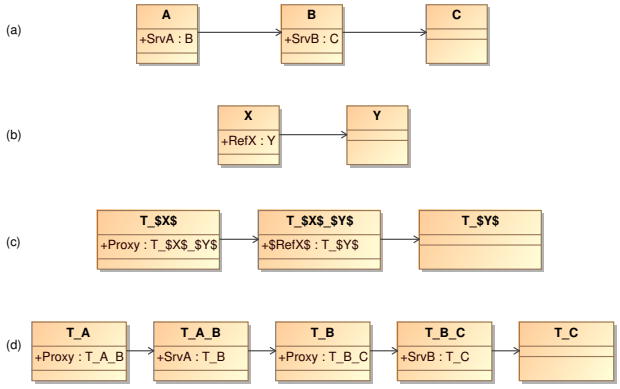

Figure 6. An example of a merging transformation involving four models: (a) source, (b) pattern, (c) template and (d) target.

Consider the following trivial example, in which, wherever one class refers to another by holding a pointer attribute, we assume there is a requirement to incorporate a proxy controlling indirection via that reference.

In figure 6, there are four models. Model (a) it the source model. Model (b) is the pattern model, and expresses the facts we are interested in all cases in some source model, where an object, no matter what its class, plays role X because it has an attribute that plays role RefX as a result of pointing to an object of a class playing role Y. In other words, model (b) matches any reference from one class to another. When the matching process is performed on the source model (a), the result is two matches. These associate roles X, RefX and Y in (b) with instances in (a), which leads

to the labelings (X⇒A, RefX⇒SrvA, Y⇒B) and (X⇒B, RefX⇒SrvB, Y⇒C) respectively.

In the substitution step, the template model (c) is used for each of these matching instances in turn. Each time, the information from the role-filling entities in the instance is substituted to construct model elements in the target (d). The first match results in construction of the target classes T_A and T_B, derived from (a) and identifies the need for a new proxy class T_A_B, which has no progenitor in the source; all the information about it is derived from the template, except for the reference name and type, which are generated from the role correspondences of RefX and Y. The same process is repeated for the second match, and yields T_C and T_B_C. However, although the template references T_B, this has already been added to the target by the first match, and so the new content is merged into the class already created.

The final step in the transformation is to copy to the target any model elements that have not yet been involved in any match, either operations or attributes of classes already processed, or completely uninvolved classes; in this example there are no such actions to be done. The identification of unmatched items is performed by marking model elements as visited as the pattern matches are processed, and then iterating over the unmarked items. In this example we have been concerned simply with class names and references, but the same approach can be applied to any model elements, including attributes, methods, action and their types.

We call the process of pattern matching and template application a transformation rule. A complete transformation consists of many transformation rules, and these are prioritized to establish a required partial order. The definition of a rule may include a declaration of other rules that must, for consistency, be applied before or after the current rule, either immediately or eventually. This might be necessary, for instance, to normalize or optimize the target structure, but it may or may not be important to do this immediately before or after the rule application. We leave to the transformation engine the selection of an optimum linear order of rule application subject to the constraints given.

In terms of the semantics of the transformation, we can see the rules from a functional point of view, with each rule application yielding a new model, which then forms the input for the next rule, together with the pattern and template for that rule. This does not mean that repeating copying is required in an implementation, since any optimization with equivalent results can be applied, but it is the simplest expression of the transformational behaviour. If a declarative view is taken, the chain of rules form a series of constraints linked via intermediate working terms (see figure 7), and, again, optimization of the structure of the constraints is clearly possible.

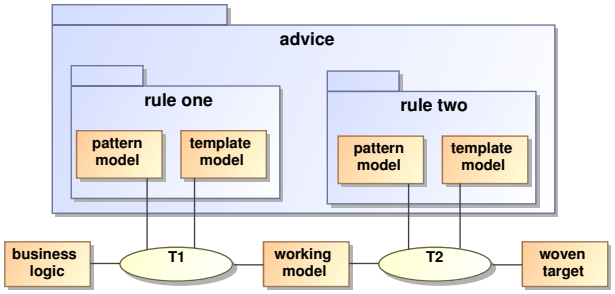

Figure 7. Linking of rules in a series of transformations.

5 Implementation issues

The approach to aspect merging described in this paper is the basis of a proof of concept pilot implementation produced by one of the authors (Liyanagama). This implementation is currently being tested with a number of scenarios, including those described here, and the details of the representations refined.

This implementation is based on the Eclipse platform and the Eclipse Modelling Framework. This environment forms a good basis for collaboration, but has some limitation because the emphasis within EMF is primarily on the static structure expressed in terms of class diagrams. The ECore model supports EClass, EAttribute, EReference and EOperation, but does not have a first-class representation of behaviour (the ECore distinction between EAttribute and EReference has also led us to use a redundant diagramming convention showing both association and reference, as being a compromise between the abstract and concrete supporting structure).

It would be possible to extend the EMF ECore to include the modelling of behaviour, but this would be a substantial piece of work in its own right. For the proof of concept implementation we have taken the approach of annotating classes with a separate algebraic representation of behaviour, in a process algebra style. The annotation covers both the behaviour of objects as autonomous entities (which we say have internal behaviour) and objects as responders to specific operation invocation (method behaviour). We believe that this approach demonstrates the requirement on the transformation mechanisms without undue additional implementation effort.

The rule structure and dependencies are expressed in a simple constraint language defined in terms of an Antlr grammar. Details of this language go beyond the aims of this paper, but it binds pattern and template to the rule and declared default behaviour (see below). There is also a separate configuration declaration that binds model names to

Eclipse resources.

6 Transformation rules for structure

Returning to the outline of the transformation process given in section 4, this section provides detail of the structural matching and rewriting process. After presenting the general mechanisms, we illustrate them by reference to the security scenarios. An almost complete representation of one of the security examples used is given in figure 8.

6.1 Specific rules

The syntactic structure of the pattern model is taken from normal UML, but the interpretation the model takes is slightly, but significantly, different from that in the normal modelling process. This is because the model is created to match cases in the base model, rather than to be the basis of an instantiation process. Model elements in the pattern are named to identify their roles in the matching process, but constraints associated with them are evaluated against potential matches. For example, the class named "X" in figure 6 (b) represents a role called "X", but could have an associated condition, such as X.name().equals("B"), which would restrict possible matches to those in which the class matched by this role has name "B". Similarly, conditions can be applied to match attribute values, or test correspondence between properties of entities in different roles. One might, for example, restrict selection to cases where two entities are associated in a specific way to an anonymous third party.

The template model is again structurally familiar from UML, but is extended by allowing any textual item to be replaced by a term construction expression that takes names or other properties from the matched roles to construct target element names. For the prototype, this string manipulation is kept very simple. Literal strings are represented directly, but material from the matching elements can be included. Role names are bracketed with "$" characters, and interpreted by reference to the matching elements for the case being processed. If they are unqualified, the name of the matching element is used, but OCL navigation expressions can be used to identify other related elements and their name used instead. In the example above, the role name "X" substitutes to "A" in the first match and "B" in the second match. The evaluation can, of course, be trivial, as in the leftmost element in figure 6 (c), where the attribute name is simply a literal.

Some pieces of syntactic sugar have been added to simplify common operations. To support transformation of operations, a notation for manipulating signature parameterization is provided, so that it is easy to specify that the aspect advice adds additional parameters to an operation signature. Another requirement is for representing systematic changes to be applied to all operations of a particular class.

Once a textual term has been constructed, it is used as a reference within the target model. If the named element does not yet exist in the target it is created; match processing is sequenced to respect containment constraints, so that classes are processed before attributes, for example, and annotations containing behavioural and other constraints last of all. If an element already exists, the features from the template are merged into it to enhance the target model.

In the course of the creation or identification of elements in the target model, tracing records of correspondence between the source and target are maintained. This is done by default by tracing name evaluation with a unique source term contribution (thus, in the example in figure 6, we have traces $A \Leftrightarrow T\_A$, $B \Leftrightarrow T\_B$, but no trace to T_A_B). In some cases, one matched element yield multiple target elements, so that there is no clear preference for one unique trace. A syntactic marker in the template is used to resolve the ambiguity, and this has proved sufficient for the test cases treated so far.

If any elements matched by the pattern are not referenced by the template model, they will not be carried forward into the target model. In the imperative interpretation, this immediately gives the effect of element deletion. In the declarative interpretation, it removes the requirement to maintain the target element, enabling it to be pruned.

Finally, a default translation is applied to any items not marked as participating in any match. This includes unreferenced classes and unreferenced attributes and operations in referenced classes. The default behaviour for the rule is declared by the rule definition, but may involve systematic renaming of elements being copied, for example. Default copying of associations between classes which have been matched draws on the tracing information to ensure that the appropriate connectivity is maintained; if the tracing information does not yield a target class, the association is discarded.

Tracking has also to be taken into account in performing the default copy step, because items being copied may include references that require renaming or retyping as a result of the transformation (see, for example, the reference in the client2 class in figure 8 (d)).

6.2 Use in the security scenarios

The selection of places where the security mechanisms need to be applied is performed by using labeling of the classes in the business logic with the names of the appropriate trust domains, introduced as marker attributes (see section 8 below for possible generalizations of this mechanism). An OCL constraint on the attributes in the two classes in the pattern representing the client and server roles

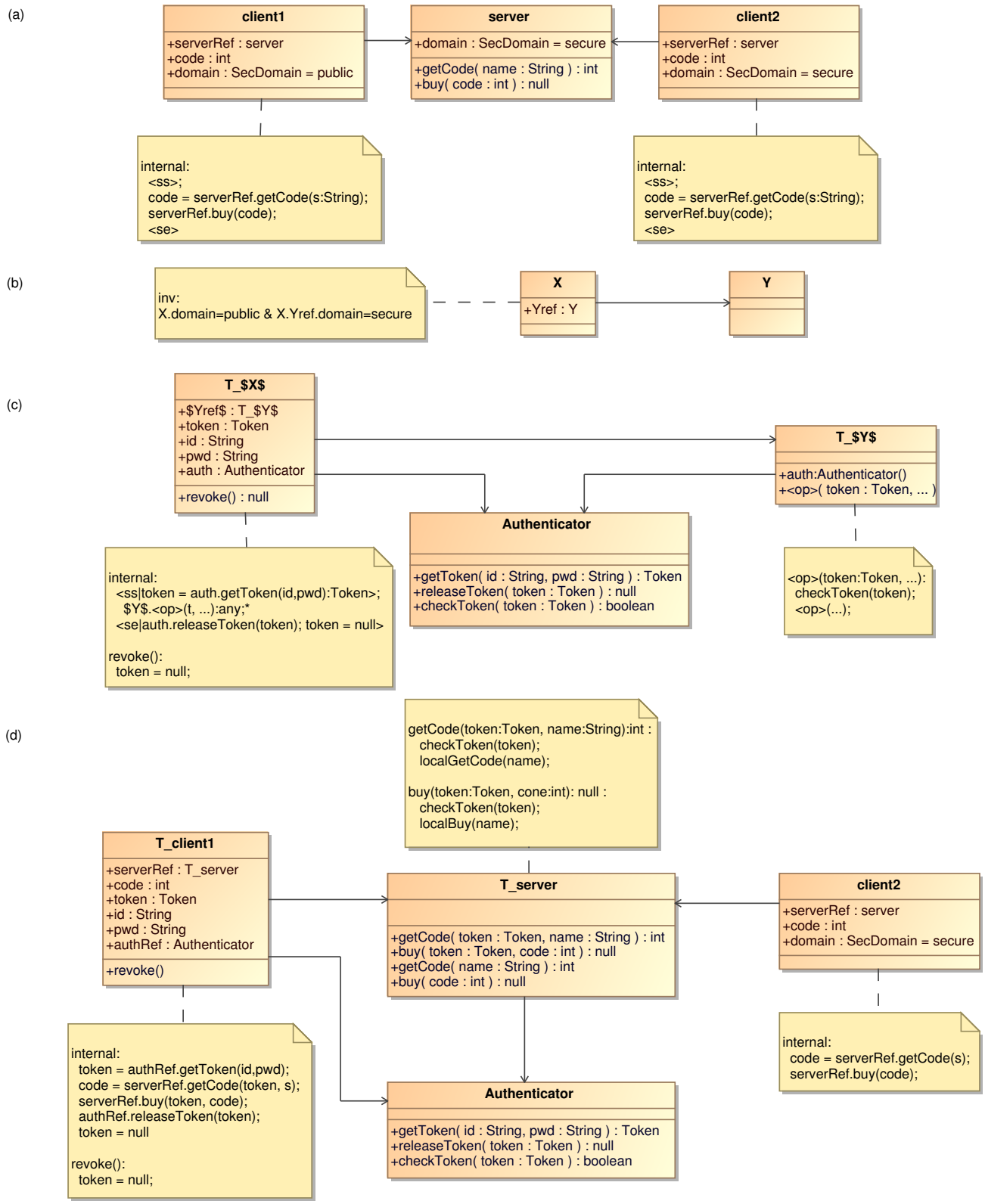

Figure 8. Security session mechanism example.

evaluates to true in the cases where there is a domain crossing from less to more secure, and hence where some insertion is required.

The manipulation of operation parameters is used in the security examples to add security specific information, such as user identity and password in the gateway mechanism and the short-lived token in the extended session mechanism.

In general, a pattern designer will need to decide whether self-references should be included or excluded, and add any necessary constraints explicitly. However, in the example given here, the domain constraint implicitly excludes self reference.

7 Transformation of behaviour

7.1 Specific rules

The first requirement in order to maintain a coherent behaviour specification is to ensure that consistency is maintained between the behaviour specification and the operations invoked by it. In order to do this, it is necessary to extract from the source model a partial call graph for invocation of operations on one class by another, rooted in the various matched roles. The call graph contains the same dependency information that would be found in a full set of sequence diagrams, but in a form that is more directly usable for steering the transformation process. Once this graph has been created, it can immediately be used to update the signatures of operations at the invocation sites to make them consistent with the target model representations.

Creating a call graph is expensive, but unavoidable if the merged model is to be a correct representation of the intended behaviour. When incorporation of a non-functional aspect requires additional information to be added to a method's parameters, for example, it is necessary to find all users of that method and modify them accordingly, recursively repeating this process until another joinpoint capable of providing the information needed is reached. In practice, the associated scaling problems are likely to limit the complexity of the aspects that can be woven with large business models.

Note that the selective application of an aspect may result in the need for support of more than one instance of, for example, a service interface. In figure 8 (d), for example, variants of operations are generated because there exist both inter and intra-domain references to the service.

The second aspect of behavioural transformation is the merging of behavioural specification in the source model and the behavioural specification in the template model. Ideally, this should be based on the same form of pattern matching as in the static structure above. However, given the pragmatic simplifications outlined in section 5, the matching mechanisms in the prototype are currently based on text matching in the annotations.

The commonest case is the expansion of the behaviour associated with operation execution, and this forms a separate named section in the behaviour of the class supporting the operation. The second significant requirement is the modification of behaviour invoking operations in other classes, and this typically involves addition of supporting behaviour to establish an aspect-specific context for the operation invocations. This requires context establishing and context closing behaviour and decoration of significant events within the bracketing formed by these. In figure 8 (c), the annotation on T_X shows the structure of a template context specification.

7.2 Use in the security scenarios

The most significant behavioural transformations are those associated with the introduction of the session-based token mechanisms. The duration of the session must be derived from some structural information in the business logic. We assume here that the scope of appropriate sessions is represented by annotations the can be recognized within the business logic.

In practice, one might use existing features, such as transaction markers to deduce context. It should, in future, be possible to perform a more detailed flow or escape analysis to determine context, but this would require a complete behavioural component in the specification. Minimizing the residual requirements for annotations on the business logic remains a research challenge.

8 Additional transformation requirements

In the scenarios analysed above, the classes in the model are annotated with their trust domains for simplicity. In the real world, however, different instances of a given class will be in different trust domains, and so the trust constraints in the selecting pattern needs to be more selective. This is a general problem, because one of the ways of simplifying the class structures in the business logic is to abstract away from the non-functional aspects.

Indeed, the reliance on static class structure in current model driven development systems does tend to lead to some lack of flexibility. Early binding of application components to specific platforms leads to rigid solutions, and speculative generation of multiple solutions increases system overhead. Late binding of a PIM to the currently available platform by just in time generation of the PSM would be an attractive option if the tool chain can be made sufficiently agile, but this implies that the transformation is driven in part by properties of instances, not type information.

These requirements lead to a need for configuration information at the instance level to be both an input and, potentially, an output of the transformation process. In the same way, availability of historical loading data at the instance level would make adaptive generation of systems that meet quality of service performance targets possible.

9 Conclusions

This paper has introduced an approach to the merging of security information and business logic that minimizes the amount of specialist transform specification required and maximizes the use of model fragments in familiar and well understood notations.

Whilst the general framework presented here could be parameterized with sources, patterns and templates that use different notations, the authors believe that there are advantages to keeping the notations used by the largest number of design practitioners the same, and encapsulating the more challenging parts of the weaving process in broadly reusable transformation rules. However, some extensions to the basic modelling language are needed; the need for template notations for constructing model element names has been demonstrated, but there are other areas. One such is the need for an ellipsis mechanism in patterns for representing a match with any arbitrary piece of model, such as an arbitrary inheritance chain or invocation path. We expect other requirements to arise from further case studies.

The current focus of our work is on the completion of our proof of concept demonstration implementation and evaluation of it using a broader set of test cases. We would expect this to involve both more complex security solutions and some investigation of other non-functional aspect requirements.

Acknowledgements

The authors would like to acknowledge the contribution to the initial development of the ideas presented here made by work done in the InterOp Network of Excellence, IST-508011, under the Information Society strand of the sixth European Framework Programme.

References

[1] J. Bézivin, G. Dup, F. Jouault, G. Pitette, and J. E. Rougui. First experiments with the atl model transformation language. In *2nd OOPSLA Workshop on Generative Techniques in the context of MDA*, 2003.

[2] J. de Lara and H. Vangheluwe. $ATo M^3$: A tool for multi-formalism and meta-modelling. In *5th International Conference on Fundamental Approaches to Software Engineering*, pages 174–188, 2002.

[3] H. Ehrig, G. Engels, H.-J. Kreowski, and G. Rozenberg, editors. *Handbook of Graph Grammars and Computing by Graph Transformation, Vol. 2: Applications, Languages and Tools*. World Scientific, Signapore, 1999.

[4] P. Epstein and R. Sandhu. Towards a UML based approach to role engineering. In *Symposium on Access Control Models and Technologies Proceedings of the fourth ACM workshop on Role-based access control*, pages 135–143, Fairfax, Virginia, USA, 1999.

[5] *ISO/IEC IS 10746-2, Information Technology - Open Distributed Processing - Reference Model: Foundations*, 1996.

[6] *ISO/IEC IS 10746-3, Information Technology - Open Distributed Processing - Reference Model: Architecture*, 1996.

[7] J. Jürjens. Model-based security engineering with uml. In *FOSAD 2004/05*. Springer Verlag, 2005. Tutorial volume, LNCS.

[8] A. Kalnins, J. Barzdins, and E. Celms. Basics of model transformation language mola. In *Workshop on Model Transformation and execution in the context of MDA, ECOOP 2004*, Oslo, Norway, 2004.

[9] G. Kiczales et al. An overview of AspectJ. In *ECOOP 2001 - Object-Oriented Programming: 15th European Conference*, Budapest, Hungary, 2001.

[10] G. Kiczales, J. Lamping, A. Mendhekar, C. Maeda, C. Lopes, J. Loingtier, and I. J. Aspect-oriented programming. In *ECOOP*, pages 220–242, 1997.

[11] G. Kiczales and M. Mezini. Aspect-oriented programming and modular reasoning. In *ICSE 2005*, pages 49–58, 2005.

[12] C. Köllmann, K. Kutvonen, P. Linington, and A. Solberg. An aspect-oriented approach to manange QoS dependability dimensions in model driven development. In *Proceedings MDEIS Workshop*, page 10, June 2007.

[13] M. Lawley and J. Steel. Practical declarative model transformation with Tefkat. In *Model Transformation in Practice Workshop, part of the MoDELS 2005 Conference*, Montego Bay, Jamaica, 2005.

[14] P. Liyanagama. A study of aspect-driven transformations to facilitate model driven developmnet. In *Interoperability for Enterprise Software and Applications: Proceedings of Doctoral Symposium of I-ESA2006*, pages 285–296. ISTE, Mar. 2006.

[15] T. Lodderstedt, D. A. Basin, and J. Doser. SecureUML: A UML-based modeling language for model-driven security. In *5th International Conference on The Unified Modeling Language*, pages 426–441, 2002.

[16] OMG. *The MDA Reference Model*, 2004. ormsc/04-02-01.

[17] OMG. *MOF Query / Views / Transformations*, 2005. ptc/05-11-01.

[18] D. Simmonds, A. Solberg, R. Reddy, R. France, and S. Ghosh. An aspect oriented model driven framework. In *The Enterprise Computing Conference (EDOC 2005)*, Enschede, Netherlands, 2005. IEEE.

[19] R. Soley and the OMG staff. The model driven architecture whitepaper, 2000. OMG document.

[20] R. J. Walker, E. L. Baniassad, and G. C. Murphy. An initial assessment of aspect-oriented programming. In *21st International Conference on Software Engineering*, Los Angeles, CA, USA, 1999.