An Aspect-Oriented Approach to Manage QoS Dependability Dimensions in Model Driven Development

Carsten Köllmann[1], Lea Kutvonen[2], Peter Linington[3], and Arnor Solberg[4]

[1] University of Duisburg-Essen, Germany.
Carsten.Koellmann@s3.uni-due.de

[2] University of Helsinki, Finland.
Lea.Kutvonen@cs.helsinki.fi

[3] University of Kent, Canterbury, UK.
P.F.Linington@kent.ac.uk

[4] SINTEF, Forskningsveien 1, 0314 Oslo, Norway.
Arnor.Solberg@sintef.no

Abstract. Model-driven development approaches commonly use an abstraction of platform specific features for improving reusability and verifiability of the core functionality models. However, the core functionality may still be tangled with features that address important dependability concerns across a design model – for example features such as security, trust and performance. These features can commonly be called Quality of Service (QoS) features. This paper presents an approach for managing several dependability dimensions. We use aspect oriented and model driven development techniques to separate and construct QoS independent models, and graph-based transformation techniques to derive the corresponding QoS specific models.

Keywords: Quality of service, Model driven development, Aspect-oriented modelling, separation of concern, integration

1 Introduction

Model driven development (MDD) shifts software development from a code-centric activity to a model-centric activity. Accomplishing this shift entails support for modelling concepts at different levels of abstraction and transforming abstract models to more concrete descriptions of software. Model driven development approaches have emphasized abstraction of platform specific features. For example, the Model Driven Architecture (MDA™) [1] specifies three abstraction levels: A Computation Independent Model (CIM) describes the environment and specifies system requirements; a Platform Independent Model (PIM) describes the parts of a solution that do not change from one platform to another; and a Platform Specific Model (PSM) includes descriptions of the parts that are platform dependent.

However, construction of distributed software systems typically involves the treatment of other dependability dimensions than just the platform dimension. The Quality of Service aspects of system design such as security, trust, performance and availability incorporate decisions about organizational policy that then need to be applied to many different business processes within an enterprise. They frequently form separate areas of concern in the design process, and techniques are needed to support this separation and ensure that unnecessary dependencies between the different sorts of design activities are avoided. Thus activities relating to extending or updating business processes should not depend in detail on the activities that ensure that enterprise-wide security or other such aspects are effectively and consistently managed. Furthermore, Quality of Service aspects of software systems are dependability concerns that tend to be tangled and cross cut the primary functionality of the system. Thus, the QoS dependability dimensions add to the complexity of software system development.

This paper describes an approach for extracting QoS concerns from the initial design process to provide QoS transparent software specifications and then introducing QoS support via suitable transformations applied to these to form QoS specific models. We use aspect oriented techniques for separating the QoS dependability dimensions. Transformation and model checking are then based on graph transformation theory.

The principal technologies, concepts and techniques of the approach are described in Section 2. First the usage of aspect oriented techniques to separate QoS dependability dimensions is described, and then the pattern matching and replacement using graphs is presented. Section 3 describes an application of these ideas to an on line shopping system to illustrate the approach. Section 4 discusses the approach and related work. Conclusions and further work are discussed in Section 5.

2 Model Driven Engineering with QoS Aspects

Model Driven Engineering (MDE) has increasingly been recognised as a key technology in providing flexibility and reducing the cost of system maintenance. It is based on the use of transformations. These transformations can be implemented within a development tool chain so that it processes design models to derive further designs that are more suited for some target design or deployment environment.

Most of the early applications of the MDE ideas were based on the need to control steps in a refinement hierarchy. The basic idea was that a transformation applied to an abstract model would generate a more specific model, based on constraints that were embodied within the transformation. The transformation is generally required to be reversible, so that round trip processes can be applied to re-incorporate changes to the more specific design into the original abstract model. This style of transformation can be applied to various steps in the refinement chain, although the most emphasis to date has been placed on the step from Platform Independent Models to Platform Specific Models, which are defined with regard to whatever concepts of computational model and platform are selected by the designers of the particular system.

As we wish to enhance this architecture by separation of QoS dependability dimensions, we need to incorporate lessons learned from known aspect technologies. Aspect-Oriented Software Development (AOSD) approaches [2][3][4] provide mechanisms for encapsulating cross-cutting features.

In particular we base the approach on the aspect oriented modelling framework (AOMDF) [5], where cross-cutting features are modelled separately as aspects and composed with the primary design model, to form integrated models. In AOMDF a design is expressed in terms of the following artefacts: i) a primary model that describes the business logic of the application, ii) a set of generic aspect models, where each model is a generic description of a crosscutting feature, iii) a set of bindings that determine where in the primary model the aspect models are to be composed, and iv) a set of composition directives that influence how aspect models are composed with the primary model.

Based on this framework we introduce an architecture that provides transparency of QoS dimensions. In this architecture the primary model is the specification of the QoS independent model of the system, while separate aspect transformations are used to provide a transparent method to refine the model and so introduce the selected set of QoS aspects. Each QoS aspect is modelled as a separate aspect model, while the primary model describes the QoS transparent model and describes the business logic of the system. In Fig. 1a the QoS independent model (QoSIM) is a system specification where the QoS aspects are transparent. The QoS specific model (QoSSM) is derived through a chain of transformations composing a selected set of QoS aspects. The approach supports transparencies with respect to a set of dependability dimensions such as security, availability, trust and performance.

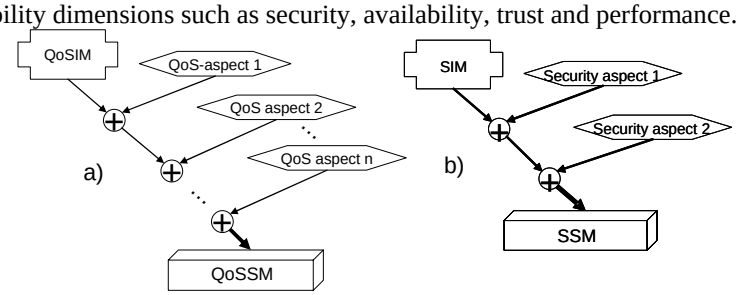

Fig. 1. a) conceptual architecture, b) transforming a security independent model to a security specific model.

A QoS dependability dimension can include several aspect models Fig. 1b shows an example where a security specific model (SSM) is derived from a security independent model (SIM) by composing it with two security related aspects.

One of the key elements in the design is the way of declaring the necessary bindings between elements in the aspect model and elements in the base application model. The elements matched may be any appropriate modelling elements, such as classifiers or relationships, and may be constrained by assertions about their required properties. Other elements in the aspect model that do not correspond directly to elements in the base model are abstracted out when forming the matching pattern. The transformation that results from this binding process will match the appropriate bound elements in the base model and use the rest of the template information either to

replace them with a refined structure or to add additional properties or supporting elements to them. The resulting model is therefore consistent with both the base application model and the aspect model being applied.

For the matching step, we utilize pattern matching and replacement using graphs for which graph transformation [7] provides a suitable formal approach. Its descriptive formalization and visual way of defining transformations supports graph modifications in an intuitive way.

Graph transformation is based on a graph grammar that includes an initial graph and transformation rules. These rules have a left hand side defining the structure that is to be modified, called a match, and a right hand side defining the target structure. The application of a rule can be guarded by preconditions, avoiding the transformation for defined contexts. Furthermore, dependencies between transformation rules can be analysed by using critical pair analysis [8] or consistency checks.

Adapting graph transformation to MDA the initial graph can be instantiated as a UML diagram and the transformation step can be interpreted as the weaving of an aspect. The application of graph transformation to different parts of UML is currently ongoing work [9], covering different issues like coherent change introduction in different diagram types, or verification of state diagrams.

Graph rules are used for the execution of the actual transformations. The syntax of the QoSIM and of the QoS aspect describing the objects and connectors of the base model are used to bind the common graph structure and rules to concrete elements.

The components that realize the chosen QoS aspect are defined at the right hand side of the rule while the join points where these are inserted are described at the left hand side.

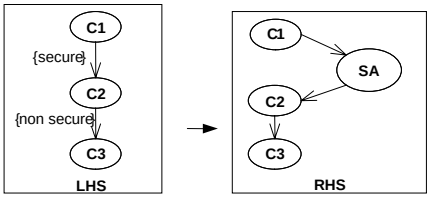

Fig. 2. Rule introducing a security component

To govern the pattern matching, tags and the categorization of model elements can be used. Fig. 2 presents a graph rule that inserts a security aspect between two components representing the join points of a given base model. The security aspect is inserted between C1 and C2, since the edge between them is categorized as a secure connection. The connector between C1 and C2 has been deleted and replaced by the new connections to the aspect.

If several rules are to be applied in sequence, care needs to be taken that there are no unintended interactions between them. Various kinds of analysis can be applied to do this. For example, critical pair analysis can be used to detect cross-links between rules and show which elements are affected, and to assess the significance of the impact. Deletion of an element by a rule when that element is referenced by a subsequent rule is likely to have serious consequences, since application of the later rule is likely to fail. Rules that exchange, or otherwise modify, elements used later on

may not cause an immediate failure, but may change the properties of the system in such a way as to undermine its intended behaviour. For example, if a performance-related aspect validates delay bounds for part of a system, and a security aspect subsequently introduces some costly encryption step within it, the performance guarantees derived earlier may be invalidated in a way that is not immediately detected. In general, guidance will be needed from the system architect to determine the priority, and hence the sequence of application, to be applied to any conflicting aspects, or conflicting features within an aspect.

3 Illustrative case study

The following case study illustrates our approach in more detail. The example system used is an online shop that includes a billing system and delivery chain, as shown in Fig. 3. A customer orders items from the online shop and pays by credit card, and the credit-worthiness is confirmed by a credit check service. The order is then processed by the shop and a corresponding order is sent to the warehouse. The Transport Service organizes and schedules the shipment to the customer.

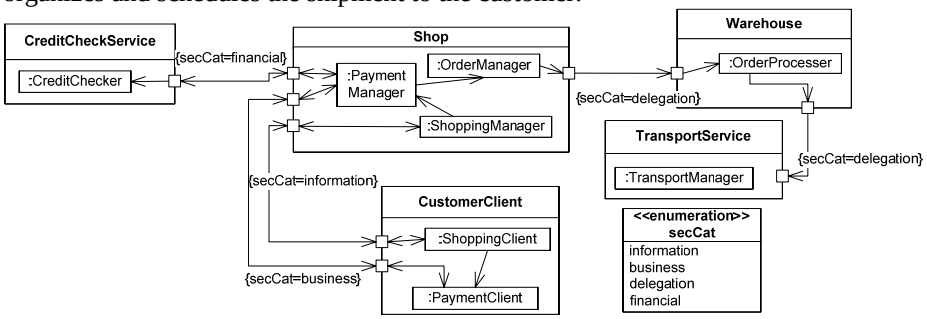

Fig. 3. Shopping application, QoS independent

The development process starts with the development of the QoS independent shopping application (Fig. 3). The primary composite structure model describes the main components, their composites and their interaction connectors. The connectors between the main components are associated with security constraints identifying the required security category needed for the actual interaction (for example {secCat=financial}). During transformation these constraints are used to identify the points where security aspects should be applied and also, which security aspect to employ.

Two security aspect models are then acquired. Aspect models will typically be specified once for a particular domain and reused across applications. Thus, aspect models will in general be acquired from an aspect model library. The authorization and encryption aspect models are shown in Fig. 4. The aspect models consist of template forms of composite structure diagrams, expressed using a template variant of the Role Based Meta-Modelling Language (RBML) [6].The aspect templates are instantiated by binding template parameters to values. In RBML, parameters are introduced by the symbol “|”. When the role binding is specified the security

independent model is composed with the aspect models according to specified composition rules and algorithms.

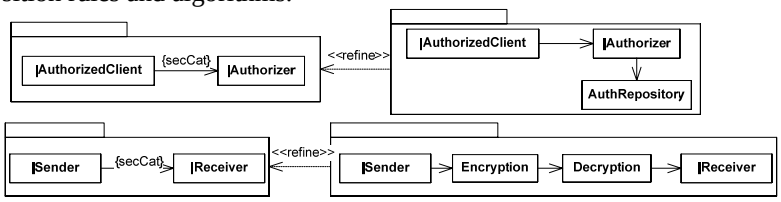

Fig. 4. Security aspects: Authorization and encryption

In our example, security categories are associated with the connectors between the main components, giving a business-oriented statement of the nature and sensitivity of the actual interactions at these points. These categories are used during the transformation to identify which security aspects need to be applied at each point. The categories are: i) *financial*: carries interactions that relate to financial status and commitment: they need to be private and non-reputable, ii) *business*: carries commitments for supply and payment; they need to be private, but carry their own credentials, so can be taken at face value, iii) *delegation*: links business partners where non-repudiation is still important, but the interactions are within the virtual organization, so that privacy is already guaranteed, and iv) *information*: the interactions carried are in the public domain.

The following composition rules are applied:

```
if (secCat = delegation or secCat = financial)
          composeAuthorizationAspect()
if (secCat = business or secCat = financial)
          composeEncryptionAspect()
```

The actual matching of components and the transformation are done by graph rules. To map the components into graph rule components, each bi-directional arrow has to be replaced by a pair of arrows, with one pointing in each direction. The types of the connectors and component and their attributes have to be mapped to node and edge types and their attributes. The graph transformation rules are derived from the aspect models. Fig. 5 shows the graph transformation rule for the encryption and authorization aspects.

Rules can be derived from a given aspect model in the following way (taking the authorization aspect as an example). First, components that have to be included in a match are identified. The primary correspondence here is with any edge having a securityCategory attribute, since this is what is referenced in the composition rules, but there are also consequential correspondences with nodes linked by such edges (giving placeholders C1 and C2). Next, the right hand side of the transformation rule is constructed. This starts by identifying those elements of the match that are to be preserved (C1 and C2 again) and then adds the necessary pattern functionality to them (here by introducing interceptor objects |AuthorizationClient and |AuthorizationServer, between C1 and C2). In general, it may also involve merging functions and attributes from the elements matched and the pattern elements, and renaming the resulting elements to reflect their new broader role. This can often result in optimizations that combine closely coupled objects (as with the replacement of C1 and AuthorizationClient by a combination called C1_name_secured). Finally, new

components are created (such as AuthRepository) and linked to the appropriate points in the structure. This optimization rule can be expressed as shown in Fig. 6.

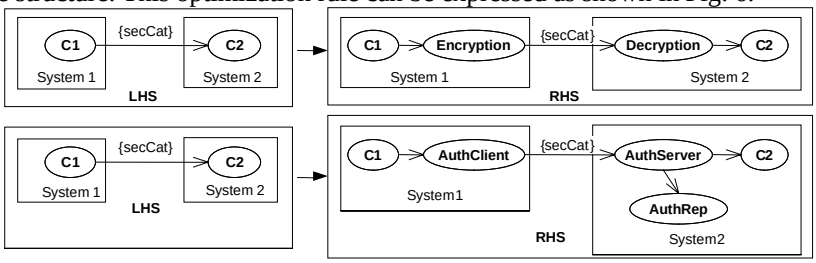

Fig. 5. Graph rules for encryption and authorization insertion

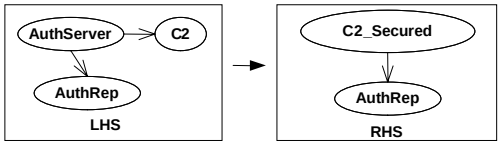

Fig. 6 Optimization rule

Executing this rule the arbitrary components C1 and C2 are preserved, while the connecting edge is deleted. To introduce the authorization aspect a client and a server component are inserted and connected to each other and the existing components. The security category of the old connection is preserved and attached to the new external connections. Applying this rule to all the matches found, the authorization mechanism is introduced for every connection between two systems. It is assumed that the transformation engine applying the rules in Fig. 5 will not do so when the structure is already equal to the structure of the right hand side, so the aspect is introduced just once for every external connection.

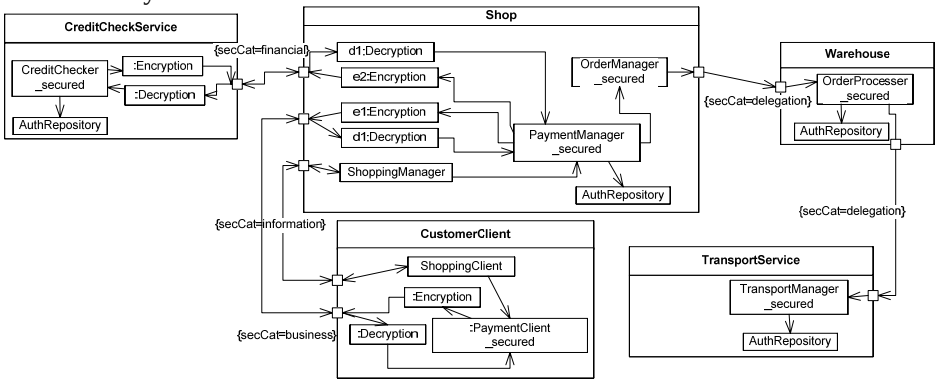

Fig. 7. Composing security specific aspects

In general nodes are merged in a graph transformation rule by deleting the old nodes while preserving the information included in their attributes. Thereafter a new node is created including this preserved information. Of course not all information has to be preserved, only that which is needed. The edges connected with the old nodes

are also deleted while the information is preserved. New edges can be generated from this information, if necessary.
In Fig. 7, the optimized security specific model resulting from the transformations described above is given. This is a view of the implementation level, where the new edge between C1 and the authorization client represents a proxy call, and is therefore implicit. The resulting modified components include both their original functionality and the added operations given by the aspect model. E.g., OrderManager_secured combines the functionality of OrderManager and AuthorizationServer.

4 Discussion and related work

One of the problems in managing a complex set of QoS dimensions is that there may be unintended interdependencies between the transformations applied. We identified earlier that this implies a need to check for conflicts and select a sequence for the transformation steps that minimizes the problems. There may be a need for the system architect to be involved in determining the safest sequence of transformations, but this intervention is itself a possible source of errors.
What is needed if the system is to be well behaved is for the later transformations in the chain to honour obligations on them not to change properties guaranteed by earlier transformation steps. This implies a piece of record keeping too complex to be performed reliably by a human user, and so some automated support is needed. There are two alternatives; either the later transformations must be analysed to prove that they do not change the properties of interest, or the model generated must include internal checks that the property achieved does remain true.
The first of these two options is hard to achieve, and, in all but the simplest cases, is likely to be undecidable. The second option seems more attractive. It requires the target of each aspect model to include not just the desired structure, but also some assertion to be generated that remains true as long as the desired properties are achieved. This is still not trivial, but is a simpler task than proving properties of the transformation steps.

There are several approaches focusing on separation of dependability concerns at the implementation level [2][11][3][10]. Even if some of the implementation level mechanisms and principles can be utilized at the model level (for example the notion of a primary model is similar to the base hierarchy in AspectJ), there are differences. Compared to AOP approaches our models are independent of any programming language and the aspects models can be implemented using a non-AOP language. Furthermore, through usage of graph transformation our approach in principal allows composition of two aspect models or even two primary models. This generality is similar to Hyperspaces [3]. Hyperspaces is an extension of subject oriented programming [10], used to achieve multi-dimensional separation of concerns at the implementation level. This is a generic separation of concern approach where any *slice* can be defined and composed with any other. The hyperslices are composed to obtain the overall system. In our approach the composition procedure depends on the model element properties specified rather than just the names of model elements,

primarily to govern the composition at a finer grained level and because not all UML model elements are named elements.
Krishna et al. [12] presents and approach for pruning unnecessary actions to better meet the QoS requirements. The pruning is based on the expected context of different products in a product line. Our approach facilitate similar pruning at the model level when deriving the QoS specific model level through fine tuning of composition. Fine tuned composition directives can be applied through the graph based pattern matching and categorization of model elements (e.g., the *secCat* categorization of connectors). Categorization can be defined based on context to avoid unnecessary actions, for example by differentiating what actions are necessary for different parts of the system for particular contexts

Gray et al. [14][15] use aspects in domain-specific models that specifically target embedded systems. Requirements, architecture and the environment of a system are captured in the form of formal high-level models that allow representation of concerns. Their research is part of Model-Integrated Computing (MIC) and extends the scope and usage of models such that they form the backbone of a development process for building embedded software systems. The work in our research can complement theirs by providing UML based approach for representing aspects.

In [16], Burt et al. explore how QoS requirements can impact decisions related to the transformation from platform-independent models in UML to platform-specific models in IDL. It complements our work but does not address how QoS requirements can be integrated in a UML specification and how they should be resolved or refined in model transformations.

5 Conclusions

Separation of concerns is recognized as a key principle to cope with complexity in software development. In this paper we have presented how to use model driven and aspect oriented techniques for the separation of QoS dependability dimensions. The approach presented provides a more flexible view than the single platform view traditionally provided in MDE frameworks like the MDA.

The introduction of QoSIM and QoSSM enables user defined QoS transparencies and supports separation of different sorts of QoS design activities from the design of the primary functionality. A chain of transformations is performed to introduce the set of dependency dimensions. This approach includes a conceptual model providing transparency of the QoS dependability dimensions. The transparencies are provided through abstractions similar to the platform abstraction. Transformations are used to carry out a refinement embodying a set of design choices related to the management of QoS aspects such as security, availability and performance.

In further research we will continue investigating how to handle conflicts when composing interrelated aspects and also the challenge of ensuring the preservation of properties discussed in the previous section. This challenge is related to the feature interaction problem, and we will base our research on results published in this area. We will also work further on ensuring consistency between behavioural and structural model views and how to achieve this using graph transformations.

Acknowledgments. The work presented in this paper has been carried out as part of the InterOP project, FP6-508011 and MODELPLEX project IST 34081. We would like to express our gratitude towards the InterOP and MODELPLEX consortium and the Commission for their valuable support.

References

1. OMG, MDA™ Guide v1.0.1, http://www.omg.org/docs/omg/03-06-01.pdf
2. Kiczales. G., Lamping, J. Mendhekar, A., Maeda, C., Lopes, C., Loingier, J., Irwin, J.: Aspect-Oriented Programming. Proc. of the European Conference on Object-Oriented Programming, Lecture Notes in Computer Science, Vol. 1241 (1997).
3. Ossher, H., Tarr, P.: Using Multidimensional Separation of Concerns to (Re)shape evolving Software, Communications of the ACM, Vol. 44, No. 10 (2001), 43-50
4. Ray, I., France, R., Li, N., Georg, G.: An Aspect-Based Approach to Modeling Access Control Concerns, Journal of Information and Software Technology, Vol. 46, No. 9 (2004), 575-587
5. Solberg Arnor, Simmonds Devon, Reddy Raghu, France Robert, Ghosh Sudipto, Aagedal.Jan Øyvind Developing Service Oriented Systems Using an Aspect Oriented Model Driven Framework In the International Journal of Cooperative Information Systems (IJCIS), Volume 15, No 4, December 2006, pp. 535-564, © World Scientific Publishing Company
6. R. B. France, D. Kim, S. Ghosh, and E. Song, A UML-Based Pattern Specification Technique. IEEE Trans. on Software Eng., Vol. 30, No. 3 (2004), 193–206.
7. Hartmut Ehrig, Gegor Engels, Hans-Jörg Kreowski, and Grzegorz Rozenberg, editors. Handbook of Graph Grammars and Computing by Graph Transformation, Vol.2: Applications, Languages and Tools. World Scientific, Signapore, 1999
8. P. Bottoni, A. Schurr, G. Taenzer, Efficient Parsing of Visual Languages based on Critical Pair Analysis and Contextual Layered Graph Transformation. Tech. Report si-2000-06, University of Rome, 2000
9. Paul Ziemann, An Integrated Operational Semantics for a UML Core Based on Graph Transformation, Logos, Berlin, 2006
10. H. Ossher, M. Kaplan, A. Katz, W. Harrison, and V. Kruskal, “Specifying subject-oriented composition,” Theory and Practice of Object Systems, Wiley and Sons, vol. 2, no. 3, 1996.
11. Batory, D. and O’Malley, S. The design and implementation hierarchical software systems with reusable components. ACM Software Engineering and Methodology (Oct. 1992).
12. Krishna, A. S, Gokhale A. S., Scmidt D. C.. Context-Specific Middleware Specialization Techniques for Optimizing Software Product-line Architectures, Eurosys’06 April 18-21. ACM 1-59593-322-0/06/0004, 2006.
13. I. Jacobson. Case for Aspects - Part I. Software Development Magazine, pages 32-37, October 2003.
14. J. Gray, T. Bapty, S. Neema, D. C. Schmidt, A. Gokhale and B. Natarajan, "An Approach for Supporting Aspect-Oriented Domain Modeling," in Proceedings of the 2nd Intl. Conference on Generative Programming and Component Engineering (GPCE'03), Erfurt, Germany, Sept. 2003
15. J. Gray, T. Bapty, S. Neema, and J. Tuck. Handling crosscutting constraints in domain-specific modeling. Communications of the ACM, 44(10):87-93, October 2000.
16. Burt et. al, “Quality of Service Issues Related to Transforming Platform Independent Models to Platform Specific Models”, Proceedings of EDOC 2002, Lausanne, Switzerland