Available online at www.sciencedirect.com
ScienceDirect
Electronic Notes in Theoretical Computer Science
ELSEVIER
Electronic Notes in Theoretical Computer Science 163 (2007) 65–79
www.elsevier.com/locate/entcs

Model Driven Development of Security Aspects

Julia Reznik, Tom Ritter

{julia.reznik,tom.ritter}@fokus.fraunhofer.de
Fraunhofer Institute FOKUS, Kaiserin-Augusta-Allee 31,
10589 Berlin, Germany

Rudolf Schreiner, Ulrich Lang

{Rudolf.Schreiner,Ulrich.Lang}@objectsecurity.com
ObjectSecurity Ltd., St John's Innovation Centre, Cowley Road,
Cambridge, CB40WS, United Kingdom

Abstract

The development of security-critical large-scale distributed software systems is a difficult and error prone process. As we learnt from practical experiences, it is especially difficult to manually define security policies, for example for access control. A human security administrator is not able to cope with the high complexity of the interactions of the application and the low level, platform specific security policy. Therefore, a new approach is needed to ease the definition of appropriate security policies. This paper shows how realisation of security aspects of a system can be automated to a great extend by applying model-driven software development techniques not only on functional properties. In the presented approach, UML models of the application's functional properties are flexibly augmented with security relevant information. Together with a high level security policy defined by the security administrator, this augmented functional model is then used in an automatic model transformation to generate the platform specific security policy. With this approach, which supports the separation of concerns in model based software engineering, we can automatically generate security-critical applications for different middleware platforms like SecureMiddleware, which is an extended implementation of the CORBA Component Model with improved support for non functional properties like security. The concepts, platforms and tools presented in the paper are currently used for the development of several large-scale and secure applications, for example for building a Virtual Air-Space Management System with strong security requirements.

Keywords: Security Aspects, MDA, Roles

1 Introduction

Model Driven Development (MDD) turned out to be most useful for the development of standard business applications. Many existing MDD solutions and tools support this domain. However, the handling of the non-functional aspects like Quality of Service, adaptability, assurance and security are mostly not sufficiently covered. In standard business applications, this is good enough; the existing MDD

doi:10.1016/j.entcs.2006.10.016

solutions are well suited to describe the structural parts of an application, e.g. components, classes or interfaces. But often these solutions lack in easy to use support of non-functional aspects.

Security, which is our main concern in this paper, is mainly implemented directly at the platform level, e.g. by configuring roles. If this is not sufficient, e.g. if more complex security policies have to be enforced, it has to be done as part of the implementation of the business code. This, of course, is a large obstacle to one of the main principles of component based software development, namely component reusability. In such cases the component implements not only its business functionality, but also a hard coded security policy. The component can only be reused if both the functional and the non-functional requirements match.

In recent years, more advanced middleware platforms became available, which were specifically tailored to meet the demanding requirements like adaptability, flexibility, robustness and security (e.g. based on CORBA Component Model). In contrast to standard business platforms, these new advanced platforms also offer support for a variety of non-functional aspects and allow separation of functional and non-functional aspects at implementation level. Some of them follow a container or capsule paradigm, which means the business functionality is implemented in a component implementation, the non-functional aspects such as access control rules are handled by the container, the component runtime environment.

Integrating MDA, UML and security is not a new approach. Jrjens defined UMLSec as an extension to UML, in order to model and verify secure systems, while we use the functional model for the generation of security policies. Our approach is more similar to Lodderstedt's SecureUML. In SecureUML, UML is extended by Role Based Access Control (RBAC). SecureUML allows an extension of UML models by access control rules and provides a direct mapping to a middleware supporting RBAC. We use similar concepts, but SecureMiddleware is not limited to a single security model like RBAC. In SecureMiddleware, we use a more flexible policy model and evaluator, and are therefore able to support other standard security models as well, for example Mandatory Access Control, or to freely define security policies as rules on attributes. While SecureUML is very well suited for standard business applications, where RBAC is the dominating security model, our far more flexible approach is required for applications in domains like defense, air traffic control or ubiquitous computing.

Since the MDA approach improves the overall software development process so significantly, it would be most beneficial to apply it also to the development of applications based on these advanced middleware platforms, with particular emphasis on the non-functional aspects. Therefore, in this paper we describe the integration of security as non-functional aspect into the overall MDA development process and tool chain, which is used to build a virtual airspace management system with strong security requirements, with respect to access control.

The paper is outlined as follows: Section 2 described the *SecureMiddleware* platform, which is our target platform for the development of secure applications. Section 3 gives an overview on how MDA principles are applied to functional and

non-functional aspects.Section 4 explains the structure of the model-based tool chain we have built to support development of secure applications in a platform independent way. Section 5 describes an example on how we apply the presented approach in the air traffic management domain. Section 6 concludes the paper.

2 Security in Component-based Applications

2.1 SecureMiddleware

Although in model-based development environment the specific properties of the target platform become less important, a reliable and efficient execution environment is still crucial for successful system development. The CORBA Component Model (CCM) [1] defines a platform which is a good choice for developing large scale distributed systems. It is based on the CORBA middleware and adds some more advanced concepts, e.g automated deployment. It also simplifies the usage of some CORBA Services. The *SecureMiddleware* platform [2] we use for realizing systems consists of an implementation of the CORBA Component Model called Qedo [3], an extended Security Framework called OpenPMF [4,9] and both are based on top of MICO, a CORBA ORB with enhanced security functionality [11].

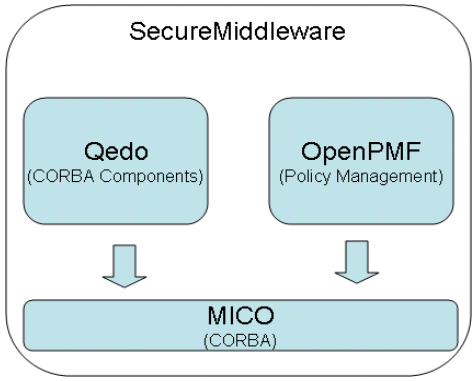

Fig. 1. SecureMiddleware platform

CORBA is well accepted by industry in mission critical application areas like Air Traffic Control, which is our target domain, because it is a reliable and mature technology and many interoperable implementations of good quality are available.

CCM enhances the Object Model of CORBA. Figure 2 depicts the features a CORBA Component can have. A component has a component interface (equivalent interface) and a component home. The equivalent interface provides operations for introspections and navigation regarding other components features; the home provides operations to manage component life cycles and must be declared for every component declaration. A component can provide a set of facets. A facet is a named port providing a specific interface. Clients of this component call operations on a facet. The facet's counterpart, a receptacle, is a named port where a specific interface can be connected to. A facet of a CORBA Component in server role can be connected to a receptacle of a CORBA Component in a client role. Receptacle

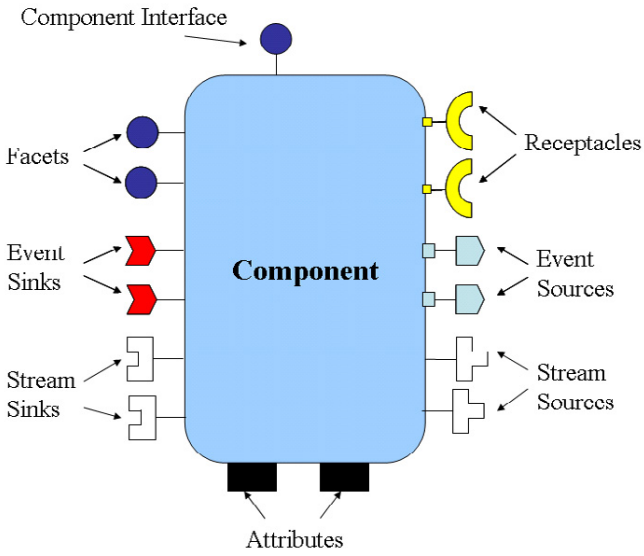

Fig. 2. Object Model of CCM

ports make dependencies to other interfaces explicit, which helps to minimize wrong configurations and run-time failures by providing type safety.

As facets and receptacles are used for operational interactions (method invocations from other components synchronously), the event sources and event sinks are used for event based interactions and message exchange (exchange event messages with other components asynchronously). An event source can publish or emit events of a certain type. Event sinks can consume events of a certain type. A similar port concept for continuous interactions (i.e. data streams) is lately introduced by the OMG to the CORBA Component Model. A stream source port produces streams of data of a specific type while a stream sink port can receive such data. Attributes can be used to configure an instance of a CORBA Component.

The CORBA Component Model has defined the container model for providing a high level of abstraction to the component implementation. It also offers the possibility to load and to unload user code (components) dynamically by installing or de-installing Homes. Depending on mechanism used by the container vendor and programming language this is realized by loading and unloading shared libraries and it requires sophisticated management of such artifacts at run-time. This is facilitated by the fact that the interfaces between the component implementation and the container are standardised. The component implementation offers a specific set of interfaces to the container allowing it to manage the component implementation. These interfaces are called Callbacks. On the other side the container offers a set of interfaces to the component implementation which enables the component implementation to make use of a certain container services. These interfaces are called Internal interfaces.

An important feature of Qedo, which makes it in particular interesting for aspect orientation, is the possibility to extend the functionality of a system or of a

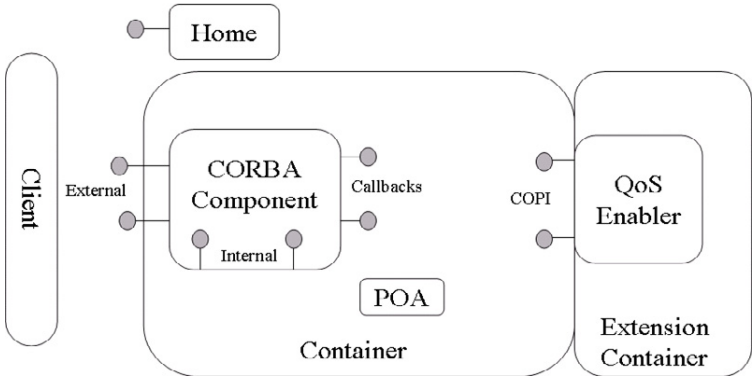

Fig. 3. QoS Enabler in extension container

component without modifyiing implementation code but by installing QoSEnablers in the run-time environment. A QoS Enabler is a specialized component that can be loaded into a specialized CCM container and is able to hook in additional functionality. Taking this approach allows usage of plain CCM mechanisms for development and deployment of QoS Enablers (see figure 3). Each QoSEnabler is responsible for a specific QoS category. In our case, a QoS Enabler is used to handle security aspects. The QoSEnabler concept is currently in final stage of standardisation at the OMG [16]. QoSEnabler may use an interception pattern to provide their functionality. To make use of it a QoSEnabler registers interception interfaces at the container. This interfaces are called Container Portable Interceptors (COPI). A more detailed description of these platfrom specifics can be found in [5].

In the SecureMiddleware we used the QoS Enablers to enforce Access Control policies. To make this possible on each relevant node a corresponding QoS Enabler is instantiated in the CCM run-time environment. The management of the security policies is accomplished by a policy management framework called OpenPMF. QoS Enablers are in contact with OpenPMF at run-time to get updates on the security policies that have to be enforced.

2.2 Policies and Policy Evaluation in OpenPMF

The OpenPMF Policy Management Framework was developed for the definition, management and enforcement of security policy in large scale distributed systems. Figure 4 shows an architectural overview of the OpenPMF framework. When the framework is initiated, the technology-neutral policy, written in a policy definition language (PDL), is loaded into a central policy repository. It is then obtained by the different systems, servers or applications and transformed into an efficient internal representation optimised for the evaluation of abstract attributes obtained from the underlying security technology and platform. At runtime, each incoming invocation triggers an evaluation process, after which the resulting decision is enforced on the particular underlying platform. In SecureMiddleware, the policy evaluation is done in an OpenPMF QoS Enabler for the components and in CORBA Portable Interceptors.

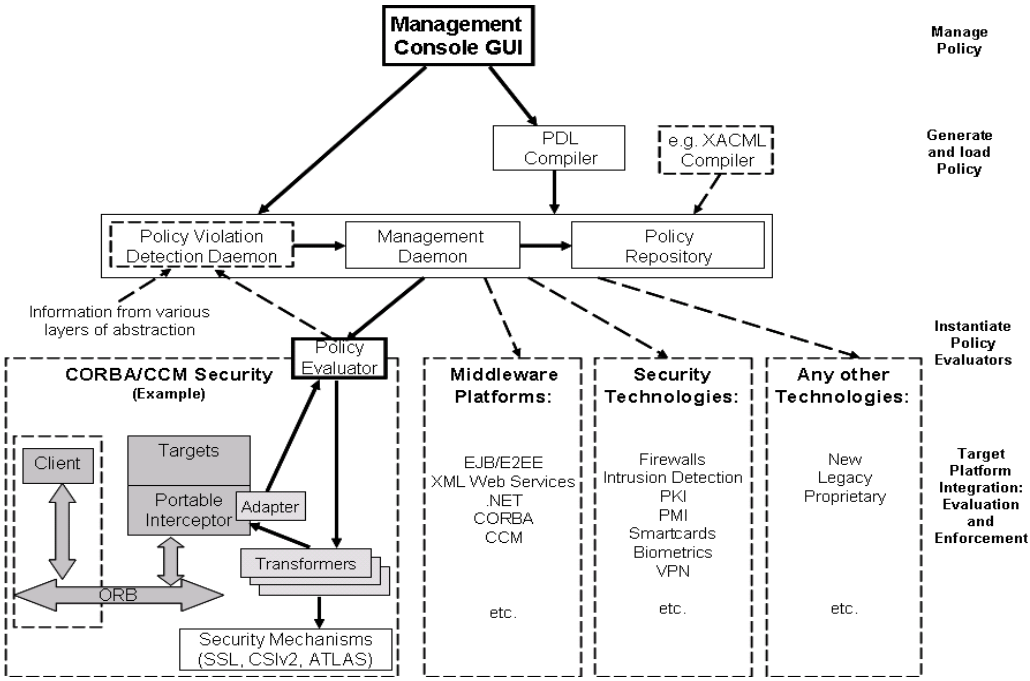

Fig. 4. OpenPMF Architectural Overview

OpenPMF is administered through the management daemon and the management GUI. In addition, the policy violation detection daemon collects relevant information from various layers of the underlying IT infrastructure and detects violations of the security policy

The current way of specifying the policy is by using our human-readable, technology-independent Policy Definition Language (PDL), which supports different security models. PDL uses concepts of the Principal Calculus [12] which theorises about principals and its two different privilege delegation relations.

PDL supports rules that are expressed in terms of requests and replies (i.e. initiating invokers, intermediaries, actions, targets etc.). Some of the features supported by the language are wildcards, multiple sets, several (arbitrary) actions, sets of clients/targets, groups and roles, and hierarchical nesting. PDL also provides advanced support for delegation. The following is a short example of a security policy definition using PDL, which allows the usage of 3 operations of the `Account` interface for a client with the name `TestUser`:

```
policy /OS [*, *] {
  policy /OS/Bank [*, *] {

     (client.name == TestUser)
     &(operation.name == deposit)
     &(target.type ==  IDL:Account:1.0) : allow;

     (client.name == TestUser)
     &(operation.name == balance)
     &(target.type ==  IDL:Account:1.0) : allow;

     (client.name == TestUser)
     &(operation.name == withdraw)
     &(target.type ==  IDL:Account:1.0) : allow;
  };
```

```
};
```

In this small example of protecting CORBA interfaces, the definition seems to be simple, because of the simplicity of the demo application. In complex real world applications the security policies very quickly become long and complex, e.g. the policy for a small CCM application is about 500 rules. While a single rule is still simple, the vast number of rules makes writing them manually and later maintaining them almost impossible. There are also dependencies between the rules, e.g. between rules for protection at a server and a domain boundary, which makes the manual rules definition even more difficult for humans. Defining security aspects of an application by writing policies for access control in PDL also requires specific knowledge of the platform specific details, to cover the internal communication of the platform, e.g. for the deployment of components. With the work we presented in this paper we want to benefit from the Model Driven Architecture to greatly reduce the complexity of writing security policies, i.e. by doing it in a platform independent way and with a higher level of abstraction.

3 Applying MDA on Functional and Non-functional Aspects

Why is the manual development of security policies so difficult, even for skilled security and middleware specialists? The first reason is the overall complexity of the systems, both of the user components and the platform internal communications. The second reason is that fulfillment of security requirements of the system is often done during the later system development phases; security is typically integrated into the resulting system in a post-hoc manner. Therefore, to make security manageable, we need to reduce the system complexity the persons in charge of security have to deal with and to improve the whole development process by providing approaches, languages and tools. Using model-centric and generative MDA approach for development of "security" systems brings following advantages:

- Abstraction and reduction of complexity: Security policies can be defined and integrated into system designs at a high level of abstraction; the human has only to make the high level decisions and the "hard work" is done by transformers.
- Well defined and structured procedures help to avoid overseeing something, which is in our experience more dangerous than making wrong decisions.
- High level models can be used to detect and correct design errors early in the deveopment process.

The main goal of our work was to provide a MDA tool chain that supports the rapid model based development process of security-critical software systems: starting from definition of "secure" system models in high-level modelling languages like UML, then transform them automatically into "secure" target system models like CCM that can be used for further steps like code generation of CCM components with security properties or plain security policy generation. In essence, apply MDA

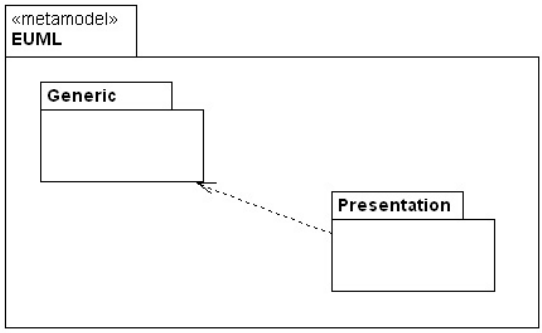

Fig. 5. Overview of the eUML metamodel

principles to functional and non-functional aspects in parallel.

The MDA tool chain and its components are described in the section 4. The idea was not to implement the whole tool chain, but to select most suitable existing modelling and other tools used during the development process with target platforms and combine these tools via model repositories in one open integrated environment. The implementation work that should be done for the tool chain is to reilize just model transformers and profiles to make it work. Artefacts like metamodels, profiles and transformers play very importand role for our tool chain realization. Metamodels provide means for management of models: all created models are stored in repositories: in our tool chain repositories are automatically generated from defined metamodels.

To support visual modelling of domain-specific aspects domain-specific languages or profiles are needed. To achieve the integration of different modelling techniques and for different modelling layers (PIM, PSM), the different repositories are interconnected together by specific model transformers, which map models to other models or to a programming or domain-specific language code.

As already mentioned, for system and security policy design we use the Unified Modeling Language (UML) [13,14] as the foundation in our work. In fact, UML is the standard for object-oriented modeling, many modeling tools support UML and a great number of developers use the language.

Since UML 2 as a whole is a language with a very broad scope and has a less precisely semantic definition (when it shall be used for automatic transformation or code generation) we have defined a subset of UML 2 together with a specialized and absolutely clear semantics. This subset we called eUML (essential UML). The eUML metamodel includes only the required UML 2 metamodel elements which formally define modeling elements, their semantic and relations. The eUML metamodel contains two main packages: Generic and Presentation as shown in figure below. The first package covers the generic modelling part (based on UML 2.0 metamodel) and the second package includes additional concepts, which define graphical modelling information of an UML element and UML diagram.

The Generic package contains the basic UML2 concepts grouped into the separate packages according to their nature. To avoid the "package merge" overhead found in the UML 2 specification, most of the generic UML 2 concepts (for example

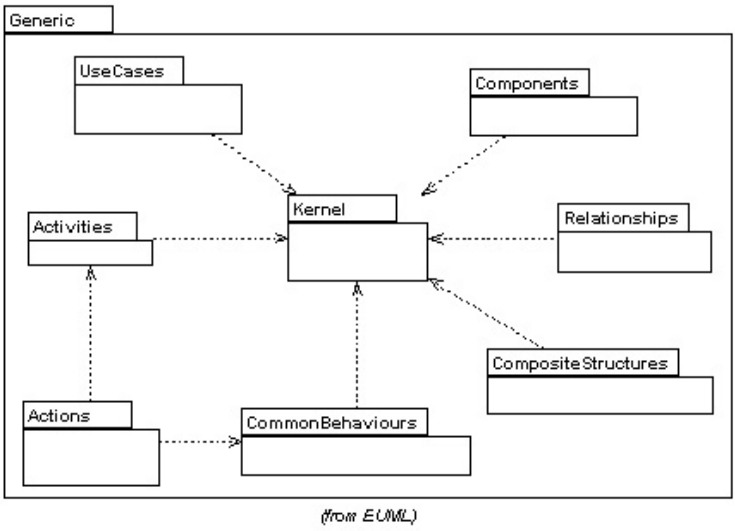

Fig. 6. Overview of the eUML Generic package content

`Class`) are represented by a single element, rather than hierarchy of multiple elements having the same name and spread out in different packages. This way seems to be most practicable and is applied in the eUML metamodel throughout.

The Generic package contains following UML2 packages: Kernel, Components, CompositeStructures, UseCases, CommonBehaviours, Actions, Activities and Relationships. Figure 6 shows the dependency relationship of the packages.

The Presentation package based on UML 2 (Generic package) and extends metaclasses like `Element` or `NamedElement` to additional graphical information such as element dimensions, positions etc. The advantage of this extention is the ability to store not only eUML model elements into repository, but also complete diagramms with all graphical information.

Similar to UML, the eUML supports user-defined UML profiles. We can extend the eUML metamodel by using profiles to customize the language for particular domain like security. For modeling our security policies we have defined a small security profile for eUML called eUMLSec. By using this profile it is possible to model the security aspect of a system additionally to the system design, for example Role Based Access Control (RBAC) [1] . A role is a job or function within a system. The role comprises all operations on a set of targets that can be executed from the user (person or system component).To reflect this concept of a role, the profile definition includes a stereotype `<<Security_Role>>` that extends the eUML metaclass Class. In the section 5 we will give an example of the secure-aware system design.

With eUML we model our target security-aware system at a high level of abstraction, the implemented transformations are responsible for the generation of security infrastructure for platform that supports RBAC and CCM. We need a well-defined metamodel for this platform to be able to transform eUML models

[1] For the sake of simplicity, we use RBAC as an example. For other policies, like Mandatory Access Control, we need additonal policies at the client side as well. Here another set of transformation is used.

(PIM) into *SecureMiddleware* models (PSM). Since the *SecureMiddleware* platform is an implementation of the CCM and OpenPMF, we have extended the CCM metamodel [1], which is already defined and standardised by the OMG. We have added security concepts like `RoleDef` and `PermittedOperation`. We call this extended metamodel *CCMSec*:

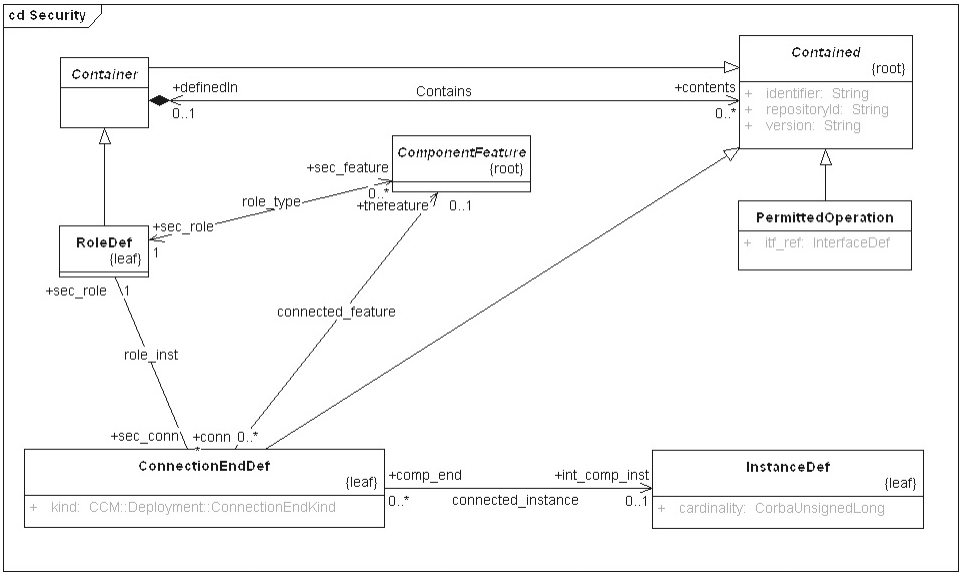

Fig. 7. CCM metamodel extension for security

The metaclass `RoleDef` inherits from the metaclass `Container` and contains `PermittedOperation` that user with this role are allowed to execute. In our approach we design also an initial configuration of component instances of an application, which is called assembly. An assembly describes the initial configuration of the application at run-time, it defines which component instance (metaclass `InstanceDef`) to use, how many and how to interconnect them to each other. We defined an assembly as UML collaboration with component instances (see example in section 5). Each component instance has ports defined by corresponding component (-type). To the ports (metaclass `ConnectionEndDef`), we added the required security information (each port is represented as "port name/role", where the "role" of the incoming port gives the required role for the invocation).

The transformation defined in our approach consists of multiple steps, because the overall security policy of a component can be separated into two distinct parts.

- **Infrastructure.** The transformation starts with the implicit generation of the security roles for the component infrastructure. This includes activities like deployment and configuration of component instances. This has to be done independently from the application security policies. In pratice, these fully automatically generated roles cover the largest part of the security policy.
- **Component Instances.** The next step is the transformation of security information modelled in component assemblies. For this we derive from every association of a role to a port of a component instance a security rule, which allows the

comunication on this component instance port for client which are in this role.

For deployment process the policies resulting from the transformation are mapped to the concrete platform, with now concrete security attributes like X.509 DN describing a component.

4 MDA Tool Chain

Our MDA tool chain has been produced for the rapid model based development of CCM based security-critical software systems based on *SecureMiddleware*. As already mentioned, it supports a platform independent modelling of systems, there is no need to model, e.g., platform-specific data types, or to decide early in the development process which particular platform to use. By using transformers, PIM models can be automatically transformed into PSM models which then can be used for further steps (e.g. C++code or security policy generation). The tool chain is a set of modeling and other development tools with adaptations to additionally support the security aspects; its architecture is shown in figure 8.

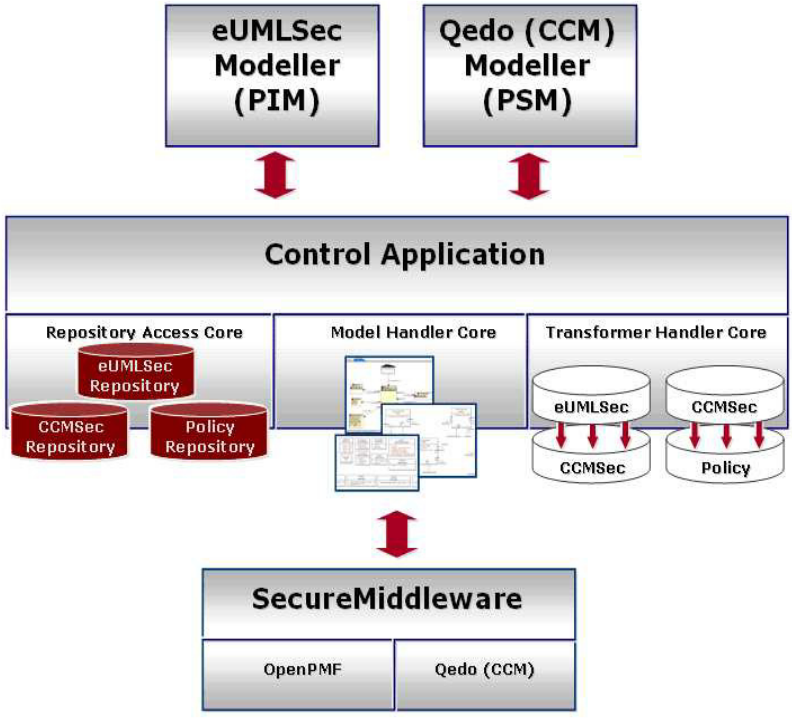

Fig. 8. Overview of the tool chain architecture

As already mentioned, we start modeling with the eUMLSec and use the eUMLSec Modeller Tool, which is a Plug-In implementation for Sparx Enterprise Architect [15]. This Plug-in is synchronized with the eUMLSec repository in both directions (load and store of eUMLSec models), including the synchronization of graphical information of the models stored in the repository. Furthermore, the eUML Plug-in

contains dialogs and wizards specific for the eUMLSec language to specify details of language elements. At this level, the model shall not contain any specific platform dependent information.

The next step in the tool chain is to transform designed model into the platform specific model, in our example the *SecureMiddleware* platform. As already mentioned in this paper, *SecurityMiddleware* is an extended CCM implementation provided by Qedo with security support provided by OpenPMF. Thus, we have two additional repositories for further transformation steps: The CCMSec repository for the management of platform specific information, and the Policy repository for management of policies defined in the platform. These repositories are synchronized with Qedo and OpenPMF and are parts of the tool chain architecture.

The heart of our tool chain architecture is a generic control application component which is used to manage and control the various components of the tool chain. The whole management of repositories, transformers and models (serialize them to the file system, incarnate them from the file system, import and export them to and from centralized shared repositories, transform them to other models) is done via the control application GUI. In the standard configuration of the tool chain control application loads three repositories: the eUMLSec, the CCMSec and Policy, and two transformers: The eUMLSec2CCMSec and CCMSec2Policy. The list of loadable tool chain components can be arbitrarily extended, we also can support other specific technologies or platforms (e.g. J2EE) and transform eUMLSec models to new platform specific models. Such extensions require at least two new tool chain components: a platform-specific repository and a transformer for eUMLSec to platform-specific transformation.

The eUMLSec2CCMSec transformer transforms an eUMLSec model from the eUMLSec repository into the CCM based specific model with security information. This transformed model will be stored in the CCMSec repository after the transformation has been done. The second transformer in the tool chain transforms CCMSec models into the Policies stored in the Policy repository. After the last transformation, the OpenPMF Policy Evaluators, here embedded in QoS Enablers and CORBA Portable Interceptors, can obtain the security policy from the policy repository and enforce it at runtime.

CCMSec models can be modified or completed (e.g. by adding deployment information or other non-functional aspects like QoS) by using the Qedo Modeller Tool implemented as an Eclipse Plug-In. The Qedo Modeller handles the connection to the CCMSec repository to propagate models into the repository and also load models from the repository and display them graphically.

From CCMSec models Qedo Modeller is able to generate the pre-generated implementation skeletons, deployment descriptors and security policy description (PDL) for the CCMSec model. After completing the business code implementation of pre-generated components the tool chain finally creates component and assembly packages and loads generated policies into the central policy repository of OpenPMF. After this step created components are immediately deployable and executable on top of the Secure Middleware platform.

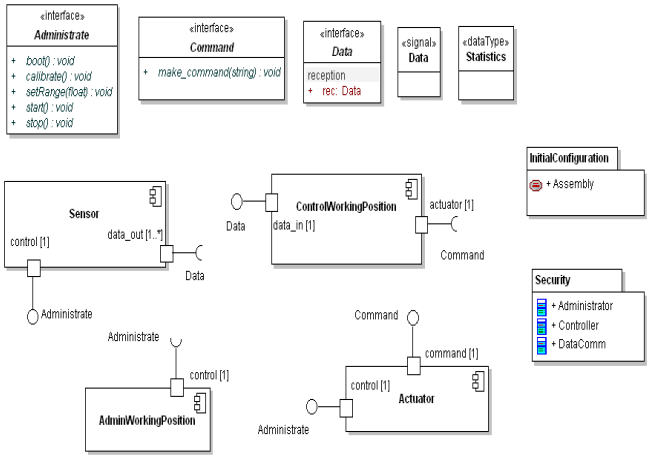

Fig. 9. Extract from the ATM system design

5 Using the Approach in the Air Traffic Management Domain

By applying MDA in our tool chain, we are able to generate automatically platform specific security policies out of platform independent models. We present a simplified and small fraction of a "Secure_ATM" (Air Trafic Management) system we are currently working on to demonstrate this. The example defines four components: Sensor, ControlWorkingPosition, AdminWorkingPosition and Actuator as outlined in figure 9.

The Sensor component represents a radar station which sends data (Data) to the ControlWorkingPosition component. The Sensor component can be controlled and managed via the interface Administrate provided by the port control. The ControlWorkingPosition collects and presents the data to flight controllers, for example. The Actuator component represents an example actuator that communicates with the ControlWorkingPosition component via interface Command. The AdminWorkingPosition component manages components via the interface Administrate. Using mentioned above profile definition for Security we defined following roles as stereotyped with `<<Security_Role>>` UML classes (figure 10).

Fig. 10. Roles overview

The Controller role includes all (_all_) operations from the interface Command, the role gives a permission to execute operations of the Command interface. The same principle is for the role Administrator that includes all operations of the interface Administrate, and the role DataComm enables Data signal based communication. It is of course also possible to select certain operations on a set of targets.

As outlined in section 3 the definition of an assembly on PIM level is done by using UML collaboration diagram. Figure 11 shows one configuration of our example application with some security information attached to it. In particular, the roles in which the component instances interact which each other are attached to the ports of the component instances. This collaboration diagram gives sufficient information for the generation of securitypolicies which are enforced by the *SecureMiddleware* platform.

The tool chain also automatically transforms the eUML component definitions into CCM specific implementation skeletons, which allows the implementation of the business code of the components. And also starting from the collaboration diagram the tool chain generates deployment descriptors and component packages which are later used for the automatic deployment of the application.

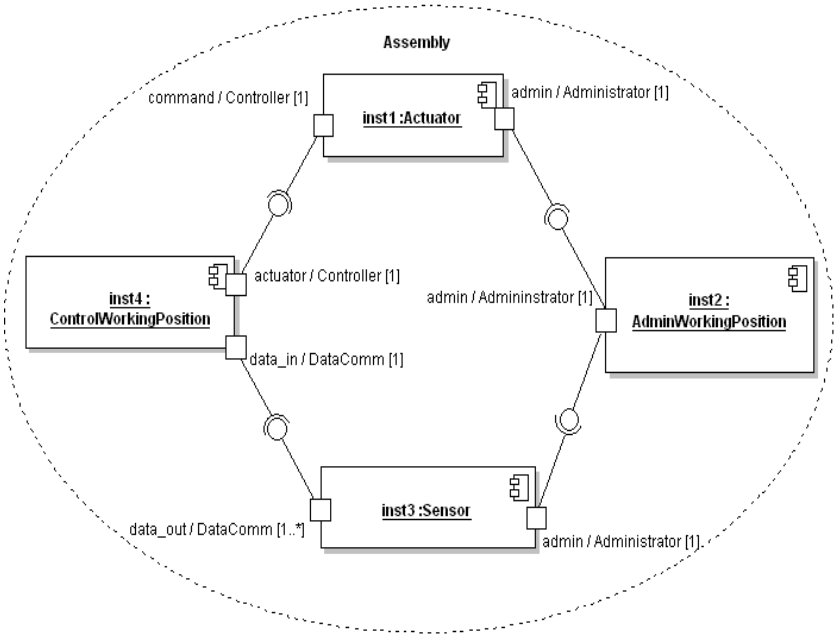

Fig. 11. Example Assembly

6 Conclusion

The Model Driven Architecture greatly improved the development of large scale distributed applications. In this paper we have described how the concepts of Model Driven Development are applied to build a security aware tool chain, which allows the platform independent definition of Air Traffic Management systems in conjunction with definition with security policies. Both, the design and the operation of such systems greatly benefited from such a security aware MDD based tool chain, i.e. more rapid development and a higher level of safety is possible. The tool-supported definition of security policies at a high level of abstraction reduces the

complexity of the task of definition of security policies. The automated transformation of high level system and security models to platform specific artifacts greatly reduces development time.

The presented tool chain is successfully used for the development of a secure prototypical Air Traffic Control visualization application as part of the European AD4 project [10]. The tool chain and the `SecureMiddleware` platform currently supports two very commonly used security models, Role Base Access Control for the invocation of operations and Mandatory Access Control for controlling information flow, but can be easily extended to other security models as well. In addition to the generation of the rules for the component interaction, our tool chain is also able to automatically generate the complex rule sets for the underlying infrastructure daemons.

References

[1] Object Management Group. "CORBA Component Model". OMG document number formal/02-06-65

[2] ObjectSecurity. SecureMiddleware Project. URL: http://www.securemiddleware.org

[3] Qedo Team. Qedo (Quality of Service Enabled Distributed Objects) CCM Implementation Web Page, URL: http://www.qedo.org, March 2006

[4] ObjectSecurity. OpenPMF Project. URL: http://www.openpmf.org

[5] Ritter, T., Lang U., Schreiner R.: Integrating Security Policies via Container Portable Interceptors, Adaptive and Reflective Middleware Workshop (ARM2005) at Middleware 2005

[6] Lodderstedt T., SecureUML: A UML-Based Modelling Language for Model-Driven Security. In UML 2002 - The Unified Modelling Language. Model Engineering, languages, Concepts, and Tools. 5th International Conference, Dresden, Germany, September/October 2002, Proceedings, volume 2460 of LNCS p. 426-441, Springer, 2002

[7] Basin D., Doser J., Lodderstedt T., Model-Driven Security: from UML Models to Access Control Infrastructures. September 4, 2005

[8] Object Management Group. Meta Object Facility Core Specification 2.0, OMG document number, formal/2006-01-01

[9] Lang, U., Schreiner, R. OpenPMF Security Policy Framework for Distributed Systems. Proceedings of the Information Security Solutions Europe (ISSE 2004) Conference, Berlin, Germany, September 2004

[10] AD4 Consortium. EU FP6 R&D project AD4 - 4D Virtual Airspace Management System, URL: http://www.ad4-project.com/

[11] ObjectSecurity. URL: http://www.mico.org/

[12] Lampson, B., Abadi, M., Burrows, M., Wobber, E. Authentication in Distributed Systems: Theory and Practice. ACM Transactions on Computer Systems 10, 4, pp 265-310, November 1992

[13] Object Management Group: OMG ptc/04-10-02: UML 2.0 Superstructure Revised Final Adopted Specification

[14] Object Management Group: OMG ptc/03-09-15:UML 2.0 Infrastructure Final Adopted Specification

[15] Sparx Systems: URL: http://sparxsystems.com.au/

[16] Object Management Group: OMG ptc/06-04-12: QoS4CCM Final Adopted Specification