MODEL-BASED SECURITY FOR DEVELOPMENT CONTEXT-AWARE MOBILE APPLICATIONS BASED UML

Salahideen M. Alhaj
CIS, IT&S, Jordan
salhaj@yahoo.com

ABSTRACT

Design and development of context-aware applications is particularly complex. Context acquisition is not an easy process. Context is changing rapidly in rich information environment. The adaptation process can be based on different types of mechanisms depending on the required dynamism and may be related to the semantics of the application. Consequently, context-aware applications need specific development mechanisms. However, developing secure context-aware applications is currently a challenging task due to the specific demands and technical constraints of mobile applications. This paper introduces model based security engineering (MBSE) approach as a framework driver for secure context-aware mobile application development (SCAMAD). Utilizing UMLsec which is an extension of unified modelling language (UML).

Keywords: Context-Awareness, Model Based Security, Secure Application, Unified Modeling Language, UMLsec.

1 INTRODUCTION

As computers become more pervasive and their functionality is more transparently integrated into homes [1] and broadband technology is introduced into residential communities [2], new applications will emerge to make everyday living easier for people [1], allow a wide range of human activities (e.g., education, entertainment, social and community gatherings, etc.) to be conducted over the Internet [2]. Such applications, which will be enabled by a ubiquitous (a pervasive) computing and communication infrastructure, will provide unobtrusive access to important information, resources and services [3]. Furthermore, these applications will access this sensitive information from many different locations [2]. Clearly, the successful deployment of such applications will depend on our ability to secure them [1]. In particular, we will have to ensure that access to information and services is granted only to authorized users, without requiring them to deal with complex security policies, burdensome access control mechanisms [3], or burdensome authentication procedures [1]. Security policies in these types of environments generally follow a static approach, where security requirements do not change over time [4]. Security requirements are assumed to be relatively static since access control decisions do not change with context, nor do they account for changing conditions in the environment [1]. For example, ways to authenticate users and protocols used to encrypt messages are fixed [4]. Additionally, the surrounding situation is rarely taken into account and security requirements mainly depend on the user’s identity (or role) [4]. The need for adaptive security (that adapts according to the situation of use) is then a requisite in order to provide fine-grained access control and to block dangerous manipulations [4]. As computing technology becomes more tightly integrated into the fabric of everyday life, it is imperative that security mechanisms become more flexible and less intrusive. To address these concerns, our research is focused on providing security services for context-aware computing environments that can adapt to changing conditions when requests are made [1]. The situation that surrounds both the requested service environment and the user’s environment is formally called context [4]. Context-awareness has been considered since a time now in designing more adaptive systems, but in the domain of security, it is rather new [4].

2 CONTEXT-AWARENESS

While most people tacitly understand what context is, they find it hard to elucidate. Previous definitions of context are done by enumeration of examples or by choosing synonyms for context [5]. Dey defines Context in [5] as: Any information that can be used to characterize the situation of an entity. An entity is a person, place, or object that is considered relevant to the interaction between a user and an application, including the user and applications themselves. Context is not limited to the physical world around the user, but also incorporates the user’s behavior, and terminal and network characteristics [6]. Context-aware computing is a new computer paradigm that determines and utilizes certain context information, such as time and location. This paradigm can provide services which the user wants if the user’s context matches context in the context-aware technology [7]. In Day’s
definition [8,9], context is divided into user context (such as user’s preferences and age), physical context (such as location and time), computer system context (such as power on/off and devices), and non-classification context. This will be used in the development of a context-aware system according to the user’s preferences.

2.1 Context-Aware Computing (ACA)

Context-Aware Computing was first discussed by Schilit and Theimer [11] in 1994 to be software that “adapts according to its location of use, the collection of nearby people and objects, as well as changes to those objects over time.” Since then, there have been numerous attempts to define context-aware computing, most of which have been too specific [12]. When we try to apply previous definitions to established context-aware applications, we find that they do not fit. We have chosen a general definition of context-aware computing [5]. A system is context-aware if it uses context to provide relevant information and/or services to the user, where relevancy depends on the user’s task [5].

![Context Models](image)

However, the goal of context-aware computing, or applications that use context, as well as computing in general, should be to make interacting with computers easier. Forcing users consciously to increase the amount of information they have to input would make this interaction more difficult and tedious. Furthermore, it is likely that most users will not know which information is potentially relevant and, therefore, will not know what information to provide [8].

2.2 Context-Aware Applications (CAA)

Similar to the problem of defining context-aware, researchers have also tried to specify the important features of a context-aware application [13, 14]. Again, these features have tended to be too specific to particular applications. Our proposed categorization combines the ideas from previous taxonomies and attempts to generalize them to satisfy all existing context-aware applications. There are three categories of features that a context-aware application can support: presentation of information and services to a user, automatic execution of a service for a user, and tagging of context to information to support later retrieval [5]. Dey in [8], Defined CAA as: Applications that use context, whether on a desktop or in a mobile or ubiquitous computing environment, are called context-aware. Context-aware applications are becoming more prevalent and can be found in the areas of wearable computing, mobile computing, robotics, adaptive and intelligent user interfaces, augmented reality, adaptive computing, intelligent environments and context-sensitive interfaces. It is not surprising that in most of these areas, the user is mobile and her context is changing rapidly. According to [15] there are two extremes when it comes to Managing Context: Context Engine (CE) or Tight Coupling (TC). When dividing the context management from the application that are going to use the context and thus protect the context in an isolated and Autonomous Systems (AS), we call it a Context Engine. The other extreme to let all contexts that an application needs be an integrated part of the application.

3 GENERAL PROCESS IN CONTEXT-AWARE SYSTEMS

Context-aware systems are usually complicated systems, and they are responsible for many jobs such as acquiring, storing, interpreting, aggregating, representation, management, reasoning, and analysis of context information for different entities with different attributes. They provide their functionalities through a collaboration process of many different components in the systems. There are various types of different context-aware systems; however, generally, a context-aware system follows four basic steps. First step is acquiring context information. Second step, the system stores acquired context data into its repository. When storing context data, what kind of data model is used to represent context information is very important. To easily use the stored context data, in third step, the system controls the abstraction level of stored context data by interpreting or aggregating context data. Finally, the system utilizes the abstracted context data for context-aware applications in many ways.

3.1 Acquiring Context Information (ACI)

Because of the diversity of context information types, context information can be acquired in many ways. Physical sensors, which are hardware devices that convert physical analogue properties into computable digital data, are used for context
acquisition. According to the types of context information, many different physical sensors can be used. However, using physical sensors is not the only way of acquiring context information. Assuming a context-aware application which recommends a music playlist based on user’s preference, weather conditions of current location, and current location of users. In this situation, user’s preference can be acquired by analyzing the user’s music play history, and the weather conditions of current location can be attained by querying a web service provided by a forecasting site. Although these context can be acquired without using physical sensors, there needs to be software modules that perform as virtual sensors. Just as physical sensors convert physical properties into context data, virtual sensors convert diverse sources into context data by analyzing them.

**What to use:** One can use MBSE within first constructs a model of the system. Then, the implementation is derived from the model, generate test sequences from the model to establish conformance of the code regarding the model. For security-critical systems, this approach allows one to consider security requirements from early on in the development process.

**Design Directions:** What context information? such as the context types, the required context quality, and the collection process. How the structure, the behavior or the parameters of the application need to be changed? What mechanisms required for the collection of the context elements? What adaptation mechanisms? What target platform? How to generate Code?]

**Procedures:** Firstly you build a Computation Independent Model (CIM). Then you build a Platform Independent Model (PIM). To create the PIM you use UML, MOF and CWM (Figure 3). And then you automatically create a Platform Specific Model (PSM) out of the PIM. The interesting thing is that you can fully concentrate the development on the functionality and behavior of the software and leave technology on the side. When you’re finished with the PIM you can transform your PIM in any proprietary platform you want (e.g. CORBA, J2EE, .NET, XMI/XML). This is the step of automatic code generation from PIM to PSM. The specific code can be for: Pervasive services, Security, Events, Transactions, Directory, and more. From there you have the base to go to every domain you like (finance, e-commerce, telecom, healthcare, transportation, space, manufacturing, and more). MDA offers you also platform interoperability, portability, platform independence and productivity. If you once have completed your PIM, you can switch to another technology by regenerating the code from it.

**Technical:** The developer creates a model and stores it in the UML 1.5/XMI 1.2 (we have UML 2.0 now) file format. The file is imported by the verification framework into the internal Metadata Repository (MDR). MDR is an XMI-specific data-binding library which directly provides a representation of an XMI file at the abstraction level of a UML model through Java Metadata Interfaces (JMI). This allows the developer to operate directly with UML concepts, such as classes, state charts, and stereotypes. [The developer can then use the aspect weaver to weave in security aspects on the model or into the code that can be generated. The resulting code can then again be analyzed for security requirements. The framework is designed to be extensible: advanced users can define stereotypes, tags, and first-order logic constraints which are then automatically translated to the automated theorem prover (ATP) for verification on a given UML model].

**MDA approach Guides:** 1. Models expressed in a well-defined notation are a cornerstone to system understanding for enterprise-scale solutions. 2. Building systems can be organized around a set of models by imposing a series of transformations between models, organized into an architectural framework of layers and transformations. 3. A formal underpinning for describing models in a set of met models facilitates meaningful integration and transformation among models, and is the basis for automation through tools. 4. Acceptance and broad adoption of this model-based approach requires industry standards to provide openness to consumers, and faster competition among vendors.

**Design Principals:** The designer has to specify how the application can adapt to the context. separating non-functional concerns, such as distribution, security, and transactions from the functional application concerns. It is not practically attainable to implement all the concerns in one single transformation. Besides splitting up transformations according to technical concerns, Designer should decompose transformations according to non-functional concerns. The designer first has to define the abstract transformations that transform the models without introducing technical details. Then he has to define more and more concrete transformations that generate concrete platform-specific models. Consequently he will first define the non-functional transformations. Then he will identify the target platform. Finally, he will specify the technical transformations. Designer has to identify the non-functional services required by the application that must be provided by the underlying middleware. Designers have to imagine all the possible adaptations according to the context. The designer can study the existing context and adaptation platforms and choose the one that best satisfies the requirements in terms of these mechanisms. The designer specifies the PIM to PSMs transformations that will transform the abstract models defined throughout the design
3.2 Storing Context Information

Most of context systems store acquired context data into their repository. Context models are closely related to context storing. Context information can be represented in many ways from very simple data model like key-value model to complex ontological model [16], and many factors such as expressiveness, flexibility, generality, and computational cost to process context-aware data depend on what kind of context model is used in the system. Figure 1 shows a set of example context models. Context data constantly acquired by sensors may require a large amount of storage space, and saving context history data may be useful for many context-aware applications. However, portable devices that participate in context-aware application have scarce resources, thus, a context-aware system should have sufficient ability to manage storage resources.

3.3 Controlling Context Abstraction Level

Context-aware system is responsible for controlling abstraction level of context information and performs context abstraction in two ways: Context aggregation and Context interpretation. Context aggregation means that the system aggregates many low-level signals (raw data) into manageable number of high level information. Context interpretation is another method that interprets context information and adds semantics. It is hard for context-aware systems to directly use the raw data provided by sensors. So, context-aware systems translate sensed signals into meaningful data so that they can understand and use context data more easily. Additionally, context-aware systems can reduce the number of context data and achieve better performance by controlling the level of context abstraction. If the context abstraction is separated from a context-aware application, then the context-aware application does not have to know the details of sensors but still can use the sensed context data by the sensors.

3.4 Utilizing Context Information for Applications

Utilizing acquired and abstracted context information as useful information for services or applications is the final step of the general context-aware system process. Context-aware systems use context information for two purposes: Context information as triggering condition and context information as additional information. Context information can be used as a triggering condition of an action. A context-aware system can use context information as action triggering condition when it wants to trigger actions if current context satisfies a specific situation. To enhance the quality of service of application, context information can be used as additional information for services or applications. These two purposes of context information usage can be combined together. We can use context information for many types of context-aware application. We present several examples of context-aware application categories. Context-Aware Personalization Providing personalized contents or information based on user’s current context information (e.g. tour guide service) Automatic Device Configuration Automatically setting up device’s configuration according to user’s current situation (e.g. screen brightness of PDA) Context-aware User Interface Optimizing user interface based on user’s current context (e.g. emphasize icons that user may select) Context-aware Suggestion Providing suggestions to users behavior based on user’s current situation (e.g. warning dangerous situation).

4 DESIGN CONSIDERATIONS OF CONTEXT-AWARE SYSTEMS

When designing context-aware system, we need to consider many aspects of context-aware systems. Context-aware systems can be implemented many ways and can have different structure, depending on what the development focus of the system is. In this section, we discuss several design considerations.

4.1 Architecture Style

Context-aware system’s representative architecture styles can be categorized into three: Stand-alone, Distributed, and Centralized Architecture. Figure 2 shows the simplified architecture diagram of each category. Characteristics, advantages, and disadvantages of each are explained below. Stand-alone Architecture a basic architecture that directly accesses sensors and does not consider context sharing of devices. This architecture can be relatively easily implemented but has limitations due to the fact that it can’t process device collaboration. This architecture is appropriate for small and simple application or domain-specific application. Distributed Architecture Context-aware systems, which have distributed architecture, can store context information in many separated devices, and there is no additional central server. Each device is independent with other devices, thus, context-aware system can ignore failure or bottleneck problem and still can continue context-aware operations. Each device manages its own context information and share context information with other devices by communicating with other devices, thus an ad-hoc communication protocol is required. However, it is hard for a device to know overall situation of every device when using ad-hoc communication protocols. Usually mobile devices...
lack of resources and computation power, so, distributed architecture are with limitations in dealing with computationally intensive applications. Centralized Architecture (Context Server) Sensors and devices are connected to a centralized context server that has rich resource and computational power, and context information is stored in a centralized server.

![Diagram](https://via.placeholder.com/150)

Figure 2: Context-Aware Systems Architecture Styles

If a device needs to get other device’s context information, the device queries the centralized server and gets the result. In this architecture, every communication is performed by querying the context server, so the communication protocol can be relatively simple than distributed architecture. By using a computationally powerful device as a centralized server, many applications which require high resources and cost can be performed. However, there is a disadvantage of this approach in that it is crucial if the centralized server fails or bottleneck problem occurs.

4.2 Handling Dynamicity

Handling dynamicity is one of important considerations to make a context-aware system possible to process sophisticated context-aware applications. Entities varying from simple sensors, resource-poor mobile devices to central server with high performance participate to process context-aware applications. At the same time, connections and disconnections of many entities may dynamically occur. A context-aware system should be able to discover and deal with dynamically changing heterogeneous entities and resources.

4.3 Privacy Protection

Privacy protection is one of the important considerations to step forward to successful implementation of context-aware systems. Context-aware systems autonomously gather information from the users, so some of the users may feel uncomfortable in that the system can use or open their information without any notice. Thus, a context-aware system should let users to express their privacy needs. Context-aware systems are responsible for protecting user’s context information from illegal accesses.

4.4 Performance and Scalability

Many operations for context-aware applications have to be processed in real time, and some context-aware systems have need of reasoning and inference functionalities which require high computational cost and resource. However, resource-poor mobile devices mainly participate in context-aware applications in most cases. Thus, context-aware system developer should consider how to manage resources for achieving acceptable performance and scalability. Also communication protocol must scale adequately to deal with a large number of communicating devices.

5 MODEL-BASED SECURITY ENGINEERING

Jürjens in [17, 18, 19, 20] developed Model-Based Security Engineering (MBSE) as a soundly based approach for developing security critical software.

One can use MBSE within first constructs a model of the system. Then, the implementation is derived from the model, generate test sequences from the model to establish conformance of the code regarding the model. For security-critical systems, this approach allows one to consider security requirements from early on in the development process.

Part of the MBSE approach is the UML extension UMLsec for secure systems development. The UMLsec extension is given in form of a UML profile using the standard UML extension mechanisms. The UMLsec can be used to specify and implement security patterns, and is supported by dedicated secure systems development processes, in particular an Aspect-Oriented Modeling approach which separates complex security mechanisms from the core functionality of the system in order to allow a security verification of the particularly security-critical parts, and also of the composed model [21]. Sommerville in [22] stress on challenges that software developers have to take care on implementing Aspect-Oriented Development (AOD), which mainly; the degree of aspects independency and the software testing process (with aspect environment) which not yet defined well. Unfortunately, the pace of required change affects developers’ ability to establish and maintain desirable levels of quality of systems. Author will focus on these subtitles; Model Driven Architecture, Model-Based Security, UMLsec since they are establishing the playing ground for successfulness approach:

5.1 Model Driven Architecture (MDA)

Model Driven Architecture (MDA) is a software development lifecycle that uses models as
its core development artifacts [23]. The idea behind MDA is to raise the level of abstractions in software engineering to develop complex applications in simpler ways [24]. The MDA approach generally separates the system functionality from the implementation details. It is a framework for Model-Driven Software Development (MDSD) defined by the Object Management Group (OMG). MDA is language, vendor and middleware neutral and therefore a very interesting topic for every software development company. The focus of MDA lies on the modeling task.

Figure 3: OMG Model Driven Architecture Model

Firstly you build a Computation Independent Model (CIM). Then you build a Platform Independent Model (PIM). To create the PIM you use UML, MOF and CWM (Figure 3). And then you automatically create a Platform Specific Model (PSM) out of the PIM. The interesting thing is that you can fully concentrate the development on the functionality and behaviour of the software and leave technology on the side. When you’re finished with the PIM you can transform your PIM in any proprietary platform you want (e.g. CORBA, J2EE, .NET, XMI/XML). This is the step of automatic code generation from PIM to PSM. The specific code can be for: Pervasive services, Security, Events, Transactions, Directory, and more. From there you have the base to go to every domain you like (finance, e-commerce, telecom, healthcare, transportation, space, manufacturing, and more). MDA offers you also platform interoperability, portability, platform independence and productivity. If you once have completed your PIM, you can switch to another technology by regenerating the code from it. There are four principles that underlie the OMG’s MDA approach:

1. Models expressed in a well-defined notation are a cornerstone to system understanding for enterprise-scale solutions.
2. Building systems can be organized around a set of models by imposing a series of transformations between models, organized into an architectural framework of layers and transformations.
3. A formal underpinning for describing models in a set of metamodels facilitates meaningful integration and transformation among models, and is the basis for automation through tools.
4. Acceptance and broad adoption of this model-based approach requires industry standards to provide openness to consumers, and faster competition among vendors.

The core standards of MDA are: Unified Modeling Language (UML) is a graphical language for visualizing, specifying, constructing and documenting the artifacts for software systems and can be used for designing models in PIM; Meta Object Facility (MOF) is an integration framework for defining, manipulating and integrating metadata and data in a platform independent manner. It is the standard language for expressing metamodels. A metamodel uses MOF to formally define the abstract syntax of a set of modeling constructs, and XML Metadata Interchange (XMI) is an integration framework for defining, interchanging, manipulating and integrating XML data and objects. XMI can also be used to automatically produce XML DTDs and XML schemas from UML and MOF models.

5.2 Model-Based Security (MBS)

Jürjens and Yu, in [25], the following framework and analysis regard the MBS. The usage of the framework as illustrated in Figure 4.

Figure 4: Tool-flow of the MBSE suite [25]

Proceeds as follows: The developer creates a model and stores it in the UML 1.5/XMI 1.2 (we have UML 2.0 now) file format. The file is imported by the verification framework into the internal Metadata Repository (MDR). MDR is an XMI-specific data-binding library which directly provides a representation of an XMI file at the abstraction level of a UML model through Java Metadata Interfaces (JMI). This allows the developer to operate directly with UML concepts, such as classes, state charts, and
It is part of the Net beans project. Each plugin accesses the model through the JMI interfaces generated by the MDR library, they may receive additional textual input, and they may return both a UML model and textual output. There are two kinds of model analysis plug-ins: The static checkers parse the model, verify its static features, and deliver the results to the error analyzer. The dynamic checkers translate the relevant fragments of the UML model into the input language for example of an ATP. The ATP is spawned by the framework as an external process; its results are delivered back to the error analyzer. The error analyzer uses the information received from the static and dynamic checkers to produce a text report for the developer describing the problems found, and a modified UML model, where the errors that are found are visualized. Besides the automated theorem prover binding presented, there are other analysis plugins including a model-checker binding and plugins for simulation and test-sequence generation.

Similarly, new adversary models can be defined. In particular, the automated translation of UMLsec diagrams to first-order logic (FOL) formulas which allows automated analysis of the diagrams using ATPS for FOL is explained in [18]. In case the result is that there may be an attack, in order to fix the flaw in the code, it would be helpful to retrieve the attack trace. Since theorem provers such as e-SETHEO are highly optimized for performance by using abstract derivations, it is not trivial to extract this information. Therefore, we also implemented a tool which Trans forms the logical formulas explained above to Prolog. While the analysis in Prolog is not useful to establish whether there is an attack in the first place (because it is in order of magnitudes slower that using e-SETHEO and in general there are termination problems with its depth-first search algorithm), Prolog works fine in the case where one already knows that there is an attack, and it only needs to be shown explicitly (because it explicitly assigned values to variables during its search, which can then be queried).

5.3 Requirements on Analysis

Jurjensin, Schreck, and Bartmann in [26] define the main goal of a security analysis AS: A satisfactory level of confidence that a given security policy or particular security requirements are fulfilled, and provide some further requirements on the security assessment process for mobile communication architectures. In particular by the high number of architectural alternatives.

That may need to be analyzed; Reproducability: The results need to be reproducible for a given architecture without risk of misinterpretation. Delegability: It is required that at least parts of the analysis can be delegated to be feasible in practice. Efficiency: The analysis must be performed in a given time-frame with a defined expectation regarding thoroughness and scope. The necessary amount of work done by a human security expert should be reducable by limiting the scope of the analysis. Parallelization: It must be possible that parts of the analysis can be performed in parallel and independently. Traceability: Results of the analysis must be traceable and give guidance how negative results can be improved on, and Expressiveness: Results must carry enough information to enable an overall risk analysis of a given architecture. To achieve these requirements, we decided to evaluate the use of a security assessment process which includes the use of models related to the given architectures and security requirements, and of automated tools to analyze these models against the given security requirements. To keep the amount of additional training bounded, we chose an approach based on the Unified Modeling Language (UML), and of the options available here is the security extension UMLsec of the UML.

5.4 UMLsec

Jurjens, Schreck, and Bartmann in [26] provide that a Part of the MBSE approach is the UML extension UMLsec for secure systems development which allows the evaluation of UML specifications for vulnerabilities using a formal semantics of a simplified fragment of the UML. The UMLsec extension is given in form of a UML profile using the standard UML extension mechanisms. Stereotypes are used together with tags to formulate the security requirements and assumptions. Constraints give criteria that determine whether the requirements are met by the system design, by referring to a precise semantics of the used fragment of UML. The security-relevant information added using stereotypes includes security-relevant information covering the following aspects: Security assumptions on the physical system level, for example the stereotype, when applied to a link in a UML deployment diagram, states that this connection has to be encrypted. Security requirements on the logical level, for example related to the secure handling and communication of data, and Security policies that system parts are required to obey. The UMLsec can then be used to check the constraints associated with UMLsec stereotypes mechanically, based on XMI output of the diagrams.
from the UML drawing tool in use [18, 27]. There is also a framework for implementing verification routines for the constraints associated with the UMLsec stereotypes. Thus advanced users of the UMLsec approach can use this framework to implement verification routines for the constraints of self-defined stereotypes. The semantics for the fragment of UML used for UMLsec is defined in [17] using so-called UML Machines, which is a kind of state machine with input/output interfaces and UML-type communication mechanisms.

6 PROPOSED DESIGN PRINCIPALS

Design and development of context-aware mobile applications is particularly complex. Context acquisition is not an easy process [28]. Context is changing rapidly in rich information environment and the adaptation process can be based on different types of mechanisms depending on the required dynamism and may be related to the semantics of the application. Consequently, Securing context-aware mobile applications development needs specific development mechanisms. Several middleware products have been defined to ease the development of context-aware applications from different point of view, but yet no solution has been specifically proposed to design secure context-aware mobile applications. UML profile allows designers to model the contexts that impact an application and the How the structure, the behaviour or the parameters of the application need to be changed? according to this context.

6.1 Design Directions

From point view of design, the following questions are essential and represent design directions:

<table>
<thead>
<tr>
<th>Question</th>
<th>Answer</th>
</tr>
</thead>
<tbody>
<tr>
<td>What context information? Such as the context types, the required context quality, and the collection process. How the structure, the behaviour or the parameters of the application need to be changed?</td>
<td>What mechanisms required for the collection of the context elements? What adaptation mechanisms? What target platform? How to generate Code?</td>
</tr>
</tbody>
</table>

6.2 Design Principals

The designer has to specify how the application can adapt to the context. Aspect-oriented programming enables developers to build applications by separating functional from non-functional aspects of the application. These aspects are combined using pointcuts and weaving [29]. To develop adaptive applications using this approach, the adaptation process is implemented as non-functional aspects. Model transformations can provide a more general sense of separation of concerns than just pure technical concerns [30] by separating non-functional concerns, such as distribution, security, and transactions from the functional application concerns. is not practically attainable to implement all the concerns in one single transformation. Besides splitting up transformations according to technical concerns, Designer should decompose transformations according to non-functional concerns. Each transformation should address only one non-functional concern so that it becomes easy to implement and to reuse [28]. This work policy leads to a set of transformations sequences that need to be applied subsequently to weave all non-functional concerns into the application model. The designer first has to define the abstract transformations that transform the models without introducing technical details. Then he has to define more and more concrete transformations that generate concrete platform-specific models. Consequently he will first define the non-functional transformations. Then he will identify the target platform. Finally, he will specify the technical transformations. Designer has to identify the non-functional services required by the application that must be provided by the underlying middleware, such as distribution, security, remote data access, deployment, etc. In the case of context-aware applications, these services are also required to be adaptive. Designers have to imagine all the possible adaptations according to the context. From security design perspective the goal is to automatically generate the model for How the structure, the behaviour or the parameters of the application need to be changed in context-aware environments? of these non-functional services (security). In one hand defined and on the other hand platform independent. Once context collection mechanisms, adaptation mechanisms and required non-functional services are identified, the designer can study the existing context and adaptation platforms and choose the one that best satisfies the requirements in terms of these mechanisms. The designer specifies the PIM to PSMs transformations that will transform the abstract models defined throughout the design direction of the MDD approach into concrete models that are specific to the chosen context platform and adaptation platform respectively. The generated variability models of the non-functional services also need to be transformed according to the services provided by the chosen underlying middleware [28].

7 RELATED WORKS

In order to effectively identify the research issue, a set of research work on related fields have been explored in two tracks:

7.1 Model-Based Security Track

Jürjens in [18] state that developing security-critical systems is difficult and there are many well-known examples of security weaknesses exploited in practice. Thus a sound methodology supporting
secure systems development is urgently needed. We present an extensible verification framework for verifying UML models for security requirements. In particular, it includes various plugins performing different security analyses on models of the security extension UMLsec of UML. Here, we concentrate on an automated theorem prover binding to verify security properties of UMLsec models which make use of cryptography (such as cryptographic protocols). The work aims to contribute towards usage of UML for secure systems development in practice by offering automated analysis routines connected to popular CASE tools. Jürjens and Fox in [19] present tool-support for checking UML models and C code against security requirements. A framework supports implementing verification routines, based on XMI output of the diagrams from UML CASE tools, and on control flow generated from the C code. The tool also supports weaving security aspects into the code generated from the models. Advanced users can use this open-source framework to implement verification routines for the constraints of self-defined security requirements. They focus on a verification routine that automatically verifies cryptobased software for security requirements by using automated theorem provers. Tolk and Turnitsa in [31] discuss the main challenges for Homeland Security applications and stress on the one of them which is the fact that the different supporting organizations, services, and nations come to the table with existing information technology, supporting established business and organization processes, and using organization-specific data models. They show how to support multi-organization processes with a federation of their heterogeneous IT solutions based on the alignment and orchestration of applications with regard to the underlying models of those solutions. While processes are orchestrated and aligned top-down, the supporting IT is migrated into a Homeland Security System-of-Systems bottom-up. Web services allow the loose coupling of participating systems and the consistent application of data engineering allows the auto-configuration of data mediation layers. This is made possible by considering first the solutions themselves, and their models (the top-down approach), and only then the application of data engineering to aligning those models (the bottom-up approach). Beres, Baldwin, and Shiu in [32] present an innovative way to assess the effectiveness of security controls where measurable aspects of controls are first captured in models and then the models are used to analyze the security data gathered from the IT environment. The aim is to lift the risk and security control assessment lifecycle from a series of people based processes to one where model based technology enhances and automates the process. Jürjens and Yu in [25] present tools to support Model-Based Security engineering at both the model and the code level. In the approach supported by these tools, one firstly specifies the security-critical part of the system using the UML security extension UMLsec. The models are automatically verified for security properties using automated theorem provers. These are implemented within a framework that supports implementing verification routines, based on XMI output of the diagrams from UML CASE tools. Advanced users can use this open-source framework to implement verification routines for the constraints of self-defined security requirements. In a second step, one verifies that security-critical parts of the model are correctly implemented in the code, and applies security hardening transformations where that is not the case. This is supported by tools that (1) establish traceability through refactoring scripts and (2) modularize security hardening advices through aspect-oriented programming. Jürjens, Schreck, and Bartmann in [26] Present a field report on the employment of the UMLsec method in an industrial telecommunications context as well as indications of its benefits and limitations, and added that, In order to make mobile communication secure, the security analysis has to be an integral part of the system design and IT management process. The perform security analysis of a mobile system architecture at a large German telecommunications company, by making use of an approach to Model-based Security Engineering that is based on the UML extension UMLsec. The focus lies on the security mechanisms and security policies of the mobile applications which were analyzed using the UMLsec method and tools. Using the UMLsec notation, the user was able to annotate his models with information regarding the security critical aspects of the system in a concise and clear way. Employing the UML profile of UMLsec, developers familiar with the extension mechanisms of the UML should have no problem to learn UMLsec quickly. Furthermore, by embedding the security analysis directly into the IT development and management process, a better understanding and clearer communication of these issues is made possible.

7.2 Context-Aware Application Modeling

In context-aware applications modelling, several context models have been defined, such as the key-value pairs [33], the object-oriented model [34], the sentient object model [35], and the models based on ontologies [36]. They proposed a high level abstraction of context information, unfortunately, without methodology for solving the model context-aware applications and their adaptation according to this context. On one hand developing reusable solutions for context acquisition, interpretation, and rapid prototyping of context-aware applications as context Toolkit [37], 5, SOCAM [38], CoBrA [39], CASS [40], and CORTEX [41]. On the other hand.
the adaptation mechanisms as: CARISMA [42], K Components [43], ReMMoc [44], OpenORB [45], CORTEX [41], and RAM [46]. Most of existing work has proposed tools to simplify the complicated development process of context-aware applications without tackling the problem of their modelling and security. These middleware and frameworks that enable context collection and that can even provide adaptation mechanisms. But at the same time they offer the research community many advantages in enabling the separation of context management and processing from the development process of applications. They play a significant role in simplifying the development of context-aware applications by implementing the mechanisms that collect and interpret the context as well as the mechanisms that adapt the application to the context, but introduce several technical details in the developed applications and reduce their portability. Sheng and Benatallah in [47], propose a UML based solution to design context-aware web services. Hendricksen and Rakotonirainy in [48] is another modelling approach that includes an extension to the Object-Role Modeling by context information. This approach allows developers to program with context at a high level without the need to consider issues related to context collection. These works are focused on context modeling and do not support adaptation aspects. The MBSE approach defines a complete process that covers all the production of secure context-aware mobile applications.

8 References


