A Model-driven Framework for Trusted Computing based Systems

Masoom Alam[1] Jean-Pierre Seifert[2] Xinwen Zhang[2]
muhammad.alam@uibk.ac.at, {xinwen.z,j.seifert}@sisa.samsung.com
[1] University of Innsbruck, AUSTRIA
[2] Samsung Information Systems America, San Jose, CA,USA

Abstract

Existing approaches for Trust Management through software alone – by their very principle – are uncompromising and have inherent weaknesses. Once the information leaves the service provider platform, there is no way to guarantee the integrity of the information on the client (or service requestor) platform. The Trusted Computing Group proposed a quantum leap in security, a hardware based "root of trust" by which the integrity of a platform – be a client or service provider can be verified. However, there is no approach for the integration of this novel but essentially straight forward concept into the distributed application development. We believe that the complexity of Trusted Computing (TC) is one of the key factors that will hinder its successful integration within the web services based distributed application realm. Model-driven techniques offer a promising approach to alleviate the complexity of platforms. This contribution has three objectives. First, we detail SECTET *– a model-driven framework for leveraging TC concepts at a higher level of abstraction. We secondly elaborate the integration of platform-independent XACML policies with the platform-specific SELinux policies. Thirdly, we share our experiences regarding the implementation results of the* SECTET *on TC based systems.*

1. Introduction

Service Oriented Architectures (SOAs) with underlying technologies like web services and web services orchestration have opened the door to a wide range of novel application scenarios, especially in the context of inter-organizational cooperation and business process integration. Applications built on web service technologies use plenty of standards like WS-Trust for trust negotiation, WS-security for the fulfilment of security requirements like confidentiality and integrity and eXtensbile Access Control Markup Language (XACML) for the specification of access policies to name a few. These open standards enable the agreement and inter -operability at the technical level, while abstracting the heterogeneity of different proprietary and legacy applications. Due to this flexibility, on the one hand, business processes having common business goals can connect their enterprise applications regardless of their platform or technology in use. However, on the other hand, this decentralized management has increased the exposure of enterprise applications and therefore, requires thorough investigations of their security implications.

One of the biggest security issues in SOAs is secure information sharing. Secure information sharing is defined as "...share but protect because of the sensitivity of the content, be it for business, personal or national security reasons", cf. [33]. There is a high-level difference between secure information sharing and retail Digital Rights Management (DRM). According to [33], "...the business model for generating revenue is much more relevant to determine these trade-offs in the case of retail DRM whereas this is much less so for (secure) information sharing. In the latter case the sensitivity of the information typically directly ties to mission objectives".

According to current best practices, secure information sharing is mostly enforced through legislation or social consent. Businesses, agencies and organizations simply pledge themselves to adhere. As a matter of fact, the information owner actually has lost control over his information once it is given away. Access control is enforced on the server side only. Information once on the client, is out of control. It is impossible to impose constraints on its usage, further dissemination to non authorized third parties as a worst case is a matter of clicks. Information owners are becoming aware of the implications of these system-inherent weaknesses and are increasingly reluctant to give away or share information. If not countered, that trend could ultimately materialize into the biggest stumbling block for the realization of the information society. This leads to the conclusion that an appropriate security concept for modern computing systems must empower the information owner with total control over his information, regardless of whether it left the owner's trust domain.

1.1. Problem Statement

Considering the state of the art and best practices of security solutions in almost any commercial deployment scenario, the need for secure information sharing is high at stakes. Increasing trust and confidence in computing systems – the building block to successful e-business, e-government or e-health – by software alone has obvious limitations, cf. [27]. A prerequisite to effective secure information sharing is the establishment of trustworthiness. This can only be achieved through TC services running on trusted platforms (e.g., platform state monitoring, run-time integrity measuring, and attestation services) (e.g., [14, 27, 31]). Trusted platforms consist of a security-enhanced operating system (e.g., SELinux [2]) and stacked on-top of trusted hardware (e.g., Intel's LaGrande [14] or AMD's Pacifica [13] extensions). With trusted platforms offering most of the low-level security controls needed for security enforcement, the systematic and correct implementation of security-critical systems remains an overly complex task, which, in many aspects, is bound to low-level technical knowledge and hence error-prone. As an example, consider the configuration of the SELinux operating system, a topic that has attracted a lot of attention lately, even in the scientific community (e.g., see [1] and [17]).

What strikes even more, is the fact that with effective enforcement of TC-based security requirements in the operating system (OS), the application layers or (even worse) the services layer still poses unresolved challenges. This gap often leads to a situation where the consideration of security concerns is postponed to the end of an engineering project, or – at best but with a similar outcome – the realization is commonly left over to developers with little or no expertise in security. Even in case of a satisfying implementation of security requirements by security experts, usually endorsing a very technical notion of security narrowed to mechanisms, algorithms and protocols, the costs of continually adapting workflows and systems to match changing business (and hence security-) requirements are very often too high. As a consequence, applications and systems remain static and optimizations are hardly feasible.

Towards addressing some of the former issues, this paper makes the following contributions:

1. We present a framework for model-driven security which bridges the gap between the specification of application- and operating system-level security objectives and their respective implementation through TC technologies. We illustrate our approach through a real life scenario from healthcare.
2. We investigate the stacking of web service security standards on top of SELinux policy model.

The rest of the paper is organized as follows. Section 2 sketches the background of our work. In Section 3, we present the conceptual foundation of the framework. Section 4 describes the extensions to the conceptual foundations of our framework. Section 5 elaborates the salient features of the implementation. Section 6 summarizes related work and finally, in Section 7, a conclusion is drawn.

2. Background

2.1. Model Driven Security

The engineering of security requirements during the system design is usually neglected. model-driven security engineering is based on model-driven software engineering and Object Management Group's related standardization initiative Model Driven Architecture (MDA) in so far as high-level security requirements are realized at the model-level and are kept separate from the underlying security architecture. As an engineering discipline, model-driven security engineering is concerned with the integration of security requirements in to all phases of system development like analysis, design, implementation, testing etc.

In our framework, we specialize the concept of MDA to *Model Driven Security* (MDS) by providing a framework in which low-level security requirements of a B2B scenario are modelled at a higher level of abstraction and merged with the business requirements modelled as Platform independent Model (PIM). These security enhanced PIMs are transformed to different *open* standard specifications (Platform Specific Model) which in turn configure our component based reference architecture [23].

The approach of model-driven software engineering builds upon two key concepts. The *Domain Specific Language* (DSL) helps to model concepts in specific application domains such as e-government and online health-care services. Domain specific languages are formalized using metamodels which are used to describe relationships among concepts in a domain. The *Transformation rules* take various aspects of the models in the problem domain as input and then synthesize implementation artefacts from the models of the problem domain.

2.2. Trusted Computing (TC)

The term "Trusted Computing" refers to a technology – introduced in the very months by the *Trusted Computing Group* (TCG) [4], in which PCs, consumer electronic devices, PDA's and other mobile devices are equipped with a special hardware chip called *Trusted Platform Module*

(TPM). In accordance with other security hardware extensions, cf. [13, 14], the TPM is empowered with cryptographic mechanisms to (1) certify remotely the integrity of the (application/system) software running on the device, (2) to protect I/O and storage of data inside the device and, (3) to strictly isolate the data residing inside memory from other potentially malicious applications. This practice is well designed to effectively fight against malicious code, viruses, privacy violations, etc. The reason is that current practices for fighting against malicious code and other threats purely at the software level by their very nature are uncompromising. Indeed, it has been learned from past experiences that a trusted and tamper-proof security basis cannot be achieved using software based solutions alone [14, 27].

2.3. Security Enhanced Linux (SELinux)

In computer security, *Discretionary Access Control* (DAC) is an access control model in which a subject with owner permissions is capable of passing permissions to any other subject. However, the fundamental weakness in the DAC model is that the ability to grant and use access creates a big security hole, where by malicious software can get control of important system resources. *Security Enhanced Linux* (SELinux) – an initiative by the National Security agency (NSA) uses *Mandatory Access Control* (MAC) mechanisms that provides only such necessary accesses a program needs to perform its job — also known as the principle of least privilege.

SELinux associates an access control attribute of the form `user:role:type` to all subjects (processes) and objects (files, IPC, sockets, etc.) which is called *Security Context*. Within the security context, the `type` attribute represents the type of the subject or the object e.g. file, directory etc. The identification of subjects, objects and their access enforcement by means of types is formally known as *Type Enforcement* (TE). The `role` attribute within the security context is built upon the `type` attribute. This means that access control in SELinux is primarily enforced via Type Enforcement. Instead of directly associating a `user` with a `type`, SELinux associates a `user` with a `role` and `role` with a `type`. The `role` merely simplifies the management of users and access control is still enforced by the TE system [25].

We believe that SELinux policies are best suited for secure information sharing because MAC policies can enable continued control of the service provider, even after the release of the information. However, the current SELinux security context is limited to Type Enforcement only. Thus, we stack the XACML Policy model on top of SELinux policy model to enable continued control of the service provider, using TC mechanisms (cf. Section 5).

3. SECTET-framework – Domain Architecture

The SECTET project cluster – an engineering framework for model-driven security – facilitates the design and implementation of secure inter-organizational workflows. Based on the SOA paradigm, the objective of the SECTET-framework is to design and implement inter-organizational workflows in a peer-to-peer environment – i.e. without central control. Case studies from the domain of healthcare and e-government provided the opportunity to apply the SECTET-framework in real life scenarios [28, 20, 29, 21, 19, 9]. The framework weaves the ideas about Model Driven Architecture, Model Driven Engineering and web services standards together for an inter-organizational workflow conceptual framework, that is more than the sum of its parts. The framework caters to the needs of a rather broad domain defined as *"Security Critical Inter-organizational workflows Scenarios"*. Due to its genericity, the SECTET-framework potentially covers a large set of component based applications from domains such as e-government, e-health, e-education etc. The modeling of security critical inter-organizational workflows with the help of domain specific language SECTET, followed by a transformation process and the execution on a web services based target architecture, are the pillars of SECTET–Domain Architecture. Subsequently, we provide a rough sketch of these building blocks of the SECTET Domain Architecture.

3.1. Domain Specific Language (DSL)

Generic security concepts like authentication, authorization etc., stay principally the same for different application domains [22]. The SECTET-framework provides a high-level repository for generic security concepts, which are adapted to specific, component-based application domains. This occurs through abstract languages defined within the SECTET-framework – defined as DSLs. Security requirements such as workflow security requirements, trust management requirements and the like are modelled using DSLs at the design level, and seamlessly integrated as security patterns, in to the business requirements models.

The SECTET-DSL is composed of two sub languages namely SECTET-UML and SECTET-PL. SECTET-UML is a UML profile for visually rendering business requirements such as data type and static security requirements such as roles and their respective hierarchies [30]. The UML profiles devise an extension mechanism for building UML models in specific domains and, is a collection of such extensions that together describe some particular modelling problem and facilitate modelling constructs in that domain. In this way, syntax is provided for constructs that do not have a UML notation.

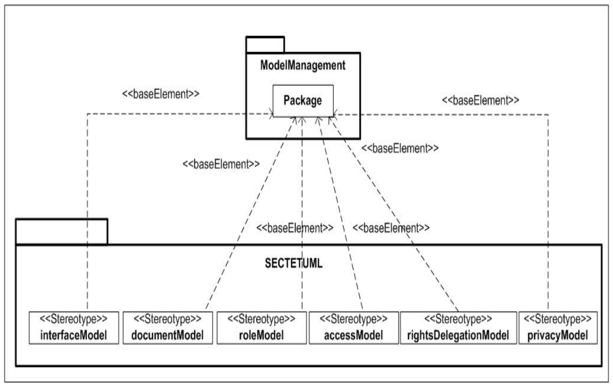
ModelManagement
Package
<<baseElement>>
SECTETUML
<<Stereotype>> interfaceModel
<<Stereotype>> documentModel
<<Stereotype>> roleModel
<<Stereotype>> accessModel
<<Stereotype>> rightsDelegationModel
<<Stereotype>> privacyModel

Figure 1. Virtual metamodel for the stereotypes of UML Packages within SECTET-UML (adopted from [5])

Description Methodology for SECTET-UML: Figure 1 and 2 shows the virtual metamodel for the stereotypes defined by SECTET-UML. The virtual metamodel is expressed via a class diagram. Each stereotype plays the client role in a dependency relationship with the UML meta class that it extends. These dependencies are stereotyped `<<baseElement>>`. Each stereotype is expressed via a classifier box, even though a stereotype is not a classifier. The keyword `<<baseElement>>` does not represent a stereotype itself, it is simply a notational marker for the underlying stereotype metaclass [5]. SECTET-UML extends the following standard UML packages.

- `Core`
- `ModelManagement`

Description: Figure 1 shows the virtual metamodel for the stereotypes used for UML packages within the SECTET-UML. For example, the UML package with the stereotype `<<documentModel>>` contains a UML class diagram which defines the data type view for the documents travelling between the partners in an inter-organizational workflow (see Fig. 3 for an example instance). In addition to exchanged document data types, it also provides an abstract view of the attributes and the resources and their relationships in the form of associations. Within the Document Model, the UML class with the stereotype `<<actor>>` hosts exclusively the attribute set that are common to all the actors within the system and all actor classes specializes the actor class (cf. Fig 2). The attribute with the stereotype `<<key>>` is used as a primary key and is used during navigation to locate a particular object of the entity. The interface with the stereotype `<<external>>` contains external functions which are used to map the requestor to its internal representation in the Document Model.

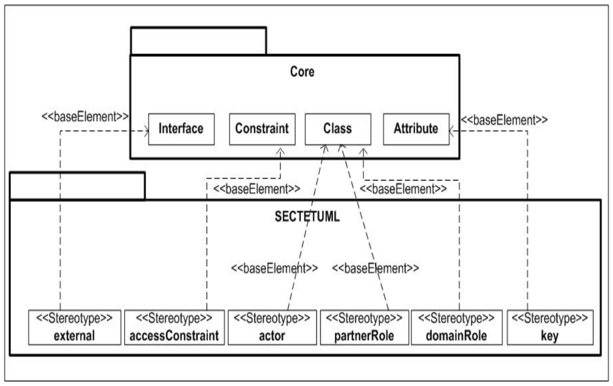

Figure 2. Virtual metamodel for the stereotypes of UML `Interface`, `Attribute`, `Constraint`, `Class`

The UML package with the stereotype `<<interfaceModel>>` defines the Interface Model which contains an abstract set of (UML-) operations representing services the component offers to its clients (cf. Fig 1). The types of the parameters are either basic types or classes in the Document Model. Additionally, pre- and post-conditions (in OCL style) may specify the behaviour of the abstract services (see Fig. 5a for an example instance).

The UML package with the stereotype `<<roleModel>>` defines roles having access to the services in the form of a UML class diagram (cf. Fig 1). The package with the stereotype `<<accessModel>>` contains the permission assignment constraints which are attached to the services defined in the Interface Model. Within the Role Model, classes with the stereotype `<<partnerRole>>` represents different partners in an inter-organizational workflow. The classes with the stereotype `<<domainRole>>` identifies the roles specific to the domain of each `<<partnerRole>>` (cf. Fig 2). In this way, a classification is made between the roles assigned to the partners in an inter-organizational workflow and entities within those partners (see Fig. 5b for an example instance).

In order to model dynamic security requirements, SECTET-UML is combined with a predicative language called SECTET-PL. SECTET-PL [7, 18, 20] – a predicative language in Object Constraint Language (OCL) style [26] – is tightly integrated with the SECTET-UML. Using SECTET-PL predicates, positive and negative permissions can be specified in the Access Model with respect to any UML class diagram. These dynamic security requirements can be transformed to any middle-ware, object-oriented se-

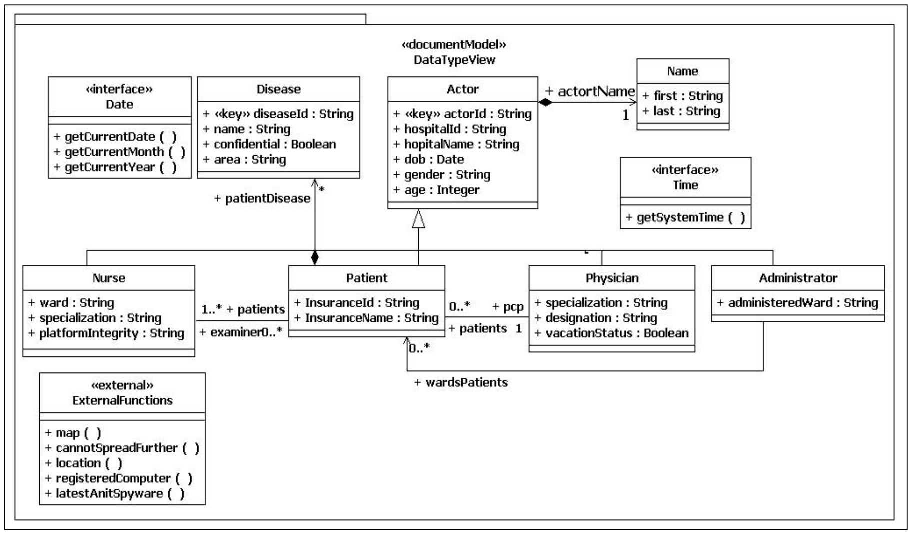

Figure 3. Sample Document Model (taken from [7])

curity platform.

In this paper, we extend the SECTET-DSL for modeling advanced access control scenarios from the domain of trusted computing.

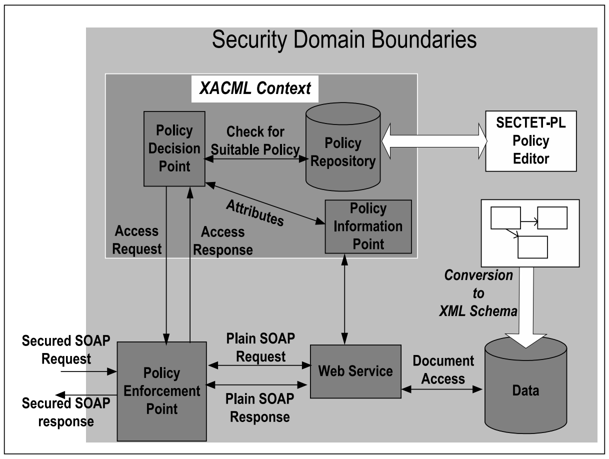

Figure 4. Target Architecture (taken from [20])

3.2. Target Reference Architecture (TA)

The Reference Architecture implemented as XACML dataflow model provides the backbone and the enabling technology for the artefacts defined at the model-level. The component employs various web services standards such as WSDL, SOAP and XACML etc. The *Policy Enforcement Point* (PEP) is an application level gateway that intercepts each incoming and outgoing message. At the service provider end, this application level gateway intercepts inbound (and outbound) SOAP requests. It first authenticates the service requester either by itself (if requester belongs to its own domain) or forwards the request to the corresponding domain. We assume that domain information is part of the service request (for information about PEP authentication handling mechanisms please see [35]). The PEP generally applies a series of security-related processing steps to the message structure in order to extract security tokens, check the signatures and decrypt message elements and to coordinate the interaction with supporting security components. In case of a successful authentication the PEP assigns a role to the request according to the credentials provided and queries a *Policy Decision Point* (PDP) to allow (or deny) access to web services (cf. Fig 4). The PDP in turn selects the applicable policy from a Policy Repository using RBAC and decides on the result of the access query.

The Policy Decision Point is primarily responsible for making authorization decisions with the help of XACML policies. After a suitable policy is found, the PDP asks the *Policy Information Point* (PIP) for attribute values necessary for policy evaluation. The PIP gathers the attributes locally or queries the appropriate partner. We use the term *Front End Authorization* for the access control performed by the PDP at the service provider end. In case authentication could not be performed by the service provider, the request is forwarded to the corresponding domain and the role assigned subsequently. In either case, the partner possessing the attributes initially receives an SAML attributes request query. These requests present a set of identifying credentials that identify the service provider to the attribute authority and include the definition of the requested web service by the service requester at the service provider end as well as credentials of the service requester himself. The attribute authority uses this information to identify and evaluate potential *Attribute Release Policies* (ARP) attached to protected attributes (i.e. to whom these attribute can be released and under which condition). If the requested attributes are protected by an ARP, the PEP queries the PDP for an authorization decision. The PDP evaluates the ARP and returns the result.

Finally, the PEP either gives the attributes to the requesting party (service provider) in the form of an SAML attributes response or returns a negative response. The access control performed by the PDP at the attribute authority end for the release of attributes is called *Back End Authorization*.

Once the complete set of requester attributes is known to the PIP at the service provider end, all return values are formulated as XACML attributes and presented to the PDP. The PDP makes the access decision on the basis of the user attributes required for the policy execution and informs his PEP about the response.

3.3. Model Transformation and Code Generation

Transformations play a key role in the MDE paradigm and are used to generate target platform specific models from the source platform independent models [8]. A transformation converts models offering a particular perspective from one level of abstraction to another, usually from a more abstract to a less abstract view, by adding more details supplied by the transformation rules. Further, a transformation process takes as input a model confirming to a given metamodel and produces as output another model confirming to a given metamodel. Considering software and system development as a set of model refinements, the transformations between models become first class elements of the software development process. As a result, defining transformations requires specialized knowledge of the business domain and the technologies used for the implementation.

Czarnecki K. et al [12] have broadly classify the transformations into two categories: *Template-based* and *Visitor-based* transformation. In Template-based, a template usually consists of the target text containing splices of metacode to access information from the source and to perform code selection and iterative expansion. We transform the Document, the Interface, the Role Model and parts of Access, Rights Delegation and Privacy Models to the corresponding XACML policy files using template-based transformations.

In visitor-based transformations, a visitor mechanism is used to traverse the internal representation of a model and then write code to a text stream. The SECTET-PL predicates are transformed to the corresponding X-Path/X-Query expressions using visitor-based transformations.

The Transformation component is prototypically implemented as a java-based tool [20] using ANTLR [10]. The prototypical tool performs the syntax analysis of the SECTET-PL predicates and afterwards verifies the predicates against the model information specified via XMI files. After the successful syntax and semantic analysis, the SECTET-PL predicates are transformed to XACML policy files.

Model-to-model transformations based on the MDA standard MOF-QVT. The mappings translate platform independent models into platform specific artefacts targeting the reference architecture. For an in-depth account on MOF-QVT based model-to-code transformation in the SECTET framework, please refer to [22].

4. SECTET Domain Architecture Extensions

4.1. Motivating Example

In order to illustrate some of the extended SECTET-framework's functionality, we take an example application scenario from a medical domain. In our case, doctors, nurses and administrators are given restricted access to resources (or patient's data) at the hospital's main site. Take a service requestor, attempting to access the online patient records, the following steps are performed during authentication and authorization:

1. A service requestor authenticates herself to the hospital site and is assigned to a role (e.g., surgeon, nurse, general practitioner etc.).

2. After authentication, the security gateway evaluates the service requestor's eligibility for the requested resource according to his role, as well as static and dynamic constraints (e.g., access on working days only).

3. If access is granted, the security gateway attaches a policy (e.g., XACML Obligations Policy) to the information released.

4. The XACML Obligation policy will be shipped with the released object/information. The platform independent XACML Obligation policy is transformed to platform specific SELinux policies. And, by using TC key-concepts, the SELinux policy enforces constraints on the future usage of the object or information on the service requestor's platform. The correct enforcement of the SELinux policies is verified through the TC-function called *Remote Attestation* [31].

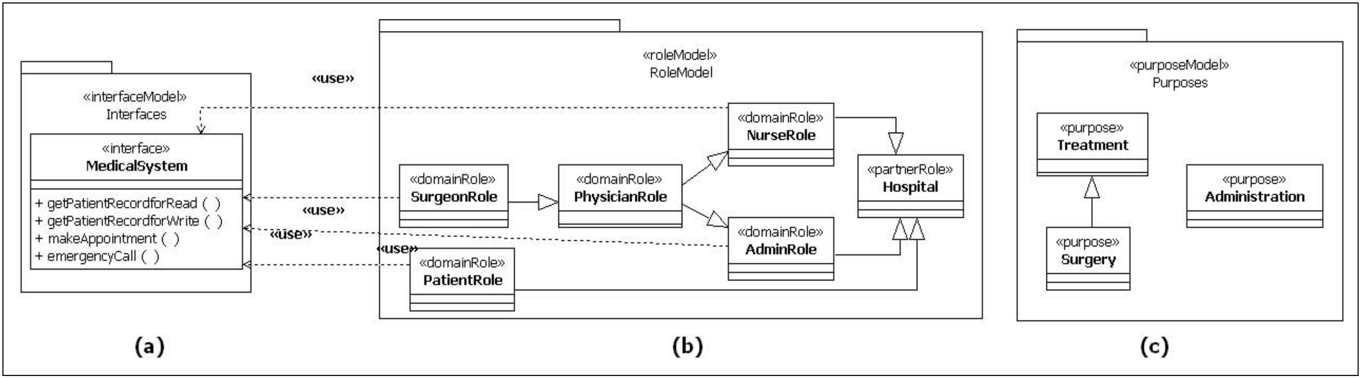

Figure 5. Sample ((a) Interface Model (b) Role Model (taken from [7])) (c) Purpose Model

4.2. Purpose Model

Being associated to the Interface View, the model type *Purpose Model* (cf. Fig 5c) represents a standard entity that defines reasons for the data collection or the intended use of the data [3]. It formulates the possible purposes in the form of a UML class diagram which is suitable for most business environments. Purposes play a key role in making access decisions in privacy and TC-aware access control management systems. In extended SECTET, a service requestor must state a purpose for the usage of the requested attributes to a service provider. Figure 5c defines three entities in the Purpose Model. The purpose `Surgery` inherits from the purpose `Treatment` which means that if some attributes can be released for the purpose `Treatment`, then the same set of attributes can also be released for the purpose `Surgery` since the latter is the subclass of the former. These purposes and their respective hierarchies are checked against the stated purpose of the service requestor using SECTET-PL described in next Section.

4.3. Predicative Specification of Access Rights in Trusted Access Model

context webService : op(x_1:T_1, x_2:T_2, ..., x_n:T_n)

perm[role$_i$] : pcondExp$_i$
...
proh[role$_j$] : ncondExp$_j$
...
oblig[role$_k$] : oactExp$_k$
...;

Figure 6. General form of Trusted Access Model

SECTET-PL [20, 7] – a predicative language in OCL-style – allows the specification of fine-grained data dependent access permissions based on roles. Originally developed with the goal of integrating aspects of authorization in use case based development [11], we use this language in the extended SECTET framework to specify permissions and actions for calling web services. We extend the Access Model for the specification of TC dependent constraints and call it *Trusted Access Model*. Like access rules, these rules are described at the model level in the predicative language SECTET-PL. The permission predicates in the *Trusted Access Model* are specified according to the general structure given in Figure 6.

The positive rule **perm**$[role_i]$: $pcondExp_i$ describes the condition $pcondExp_i$ under which some role $role_i$ is permitted to call web service `op`. The negative rule **proh**$[role_j]$: $ncondExp_j$ describes the condition $ncondExp_j$ under which some role $role_j$ is prohibited to call web service `op`. The Access Model is extended to include the obligtaion constraints. The obligation rule **oblig**$[role_k]$: $oactExp_k$ describes a sequence of actions $oactExp_k$ that some role $role_k$ is obligated to perform after the release of the requested information.

The conditions are permission predicates over the formal parameters of the web service $(x_1 : T_1, x_2 : T_2, \ldots, x_n : T_n)$. Permission predicates allow navigation through XML documents, comparison of expressions and the connection of predicates by logical operators.

The example Access Rules (ARs) in Figure 7 refer to the Document, Interface, Role and Purpose Model in Figures 3, 5a, 5b and 5c respectively. We first describe the content independent constraints followed by TC and content dependent constraints.

The *Content Independent* constraints make use of context function classes within the Document Model such as Date, Time etc. These context functions return domain specific values such as current date, system time or other system dependent values (cf. AR1). Further, the web service parameters can also be verified against some user-defined

AR1: *An administrator is allowed to check the medical records in working hours only.*

```
context MedicalSystem:getPatientRecordforRead(patID:String, patDiseaseId:String)
   perm[AdminRole]:
   Time.getSystemTime()>=8 and Time.getSystemTime()<=17;
```

AR2*: A person, identified by his or her patient number, is allowed to make an appointment only for weekdays.*

```
context MedicalSystem:makeAppointment(patID:String,phyID:String,Day:String)
    proh[PatientRole]: Day = "Saturday" or Day = "Sunday";
```

AR3*: A patient, identified by his/her patient number can read his/her own medical records only.*

```
context MedicalSystem:getPatientRecordforRead(patId:String)
    perm[PatientRole]:let pat=subject.map(Patient) in
    ph.actorId=patId;
```

AR4*: A nurse can read the medical records of every patient for her specialization and her platform should be in a known good state. But she shall not be permitted to read the medical records of the patient if patient disease is marked confidential.*

```
context MedicalSystem:getPatientRecordforRead(patId:String, patDiseaseId:String)
   perm[NurseRole]: let nr = subject.map(Nurse) in
    nr.patients.selectone(actorId=patID). patientDisease.
    select(diseaseId =  patDiseaseId and area = nr.specialization) .notEmpty() and
     nr.platformIntegrity = 1;

   proh[NurseRole]: let nr = subject.map(Nurse) in
    nr.patients.selectone(actorId = patId). patientDisease. select(diseaseId =
    patDiseaseId and confidential = true).notEmpty();
```

AR5*: A physician can modify any medical record for which he or she is the designated primary care physician for treatment purposes only. But the service requestor should not further disclose it to any third party. Moreover, the medical records should be accessible from a registered computer at the office and some mechanism should be used to prevent possible information leak.*

```
context MedicalSystem:getPatientRecordforWrite(patId:String)
    perm[PhysicianRole]: let ph=subject.map(Physician) in
      ph.patients.select(actorId=patId).notEmpty() and
      subject.confirmPurpose(ph.statedPurpose);
    oblig[PhysicianRoles]:
      subject.hoursAllowed(48);
      subject.medicalSoftwareIntegrity(1);
```

Figure 7. Sample Access Rules

values (cf. AR2).

The *Content Dependent* constraints depend on the internal representation of the calling actor (service requestor). The underlying XML documents are provided by the parameters of the web services on the one side, and by the special construct `subject.map(T)` on the other side which is supported through a library of *external functions*. The Document Model hosts the library of external functions in the interface `ExternalFunctions` stereotyped with `<<external>>`. This stereotype indicates that the corresponding interface is not transformed to XML schema but refers to the security infrastructure in order to verify a certain relationship between the caller of the web service and a particular element of the Document Model (e.g. `map`). The *identification variables* (e.g. `subject`) associated with these external functions distinguishes different types of callers, e.g., `subject.map(T)` allows the connection of the calling actor with his/her internal representation to the business logic enabling permissions like "*the actor has access to his/her own data*".

In AR3, an association between the calling `Patient` whose identity is passed as parameter is required. In order to provide this connection, we make use of the `subject.map` construct mapping the calling actor (in the `PatientRole`) to its internal representation in the form of entity `Patient`.

Used in some permission or prohibition expressions, the special construct `subject.map(T)` authenticates the caller of the web service (where the way how authentication is done can be freely chosen), checks if the caller is in the specified role and maps it an to an internal representation in the Document Model. In case the caller belongs to the some other domain, the `subject.map(T)` requests the attribute values that are not present locally from the corresponding domain through an attribute requesting service [9].

The *trusted computing dependent* constraints verify the integrity of the software and hardware on the service requestor platform, before an access is granted to the requested information. The integrity of the service requestor's platform is represented as attributes within the internal representation of the service requestor. For example, `platformIntegrity` is an attribute within the entity `Nurse`, whose value depends on the trust worthiness of the service requestor platform such as the trustworthiness of its memory, protected I/O, and secure ports. For simplicity, we represent the hardware integrity as one attribute – `platformInegrity`. In practise, many such domain specific attributes such as `medicalSoftwareIntegrity` etc could be augmented to the internal representation of the service requestor.

The trusted computing dependent attributes are requested from the service requestor as part of the trust negotiation, before an access is granted (see e.g. [32]). Further, these trusted computing dependent attributes are verified against some known values. In our setting, `1` represents a `KNOWN_GOOD_STATE`, `2` represents `INTERMEDIAY` and `0` represents `NOT_GOOD_STATE`. The next example shows a combination of positive, negative and trusted computing dependent constraints.

The *obligation* constraints define a sequence of actions that must be fulfilled by the service requestor once the requested information is released and reaches the service requestor platform. Basically, the obligations restrict the service requestor regarding the future usage of the released information by means of Security Enhanced Linux (SELinux) [2] policies (described in the next Section). These obligations are specified as external functions which obligate the service requestor for the verification of certain hardware/software requirements on her platform. The obligation constraints are transformed to XACML and SELinux policies by applying a series of transformation steps.

In addition to content dependent constraint, in AR5, the obligation constraints restrict the service requestor (`PhysicianRole` in this case) regarding the maximum duration, the released information can be viewed and that the integrity of the medical software should be in a "known good state".

The most important functionality of a TC-platform is to remotely certify to third parties in enciphered form, which software is running, whether malicious code has modified the corresponding software, status of the hardware components etc. This feature enables the service providers to deploy their services across geographical boundaries. However, recent research [15] shows that there are several shortcomings of this traditional way of remote attestation. For example, what is desired in a remote attestation is the behaviour of the software running, but what is attested is the fact that a particular binary is run [15]. Our approach towards addressing this problem is the attestation of all those properties included in the obligations. Since obligations defines the expected behaviour of the client platform, the client platform is restricted to report the properties included in the obligations to the data provider (cf. Section 5.3).

The permission constraint in AR5 uses the external function `confirmPurpose` which returns a Boolean value depending on the match of the stated purpose of the service requestor against the purpose set for the modification of the patient record by the corresponding patient. The operation also takes purpose hierarchy into account as described in Section 4.

4.4. Transformations

Transformations play a key role in MDE paradigm and are used to generate target platform specific models from the source platform independent models. Due to space restrictions, only a bird eye view of the overall transformation workflow is included for the sake of continuity.

In the SECTET-framework, we transformed the high-level access control specifications to XACML policies using QVT [22]. The generated XACML policies are then interpreted by the policy decision point in the target architecture. Consider an example policy in Figure 8 The policy formalizes the high-level security requirement that a physician identified by `PhysicianRole` can access the hospital site between 9 AM and 5 PM. (Note the example is given in simplified syntax).

XACML Obligations are post conditions which must be fulfilled by the service requestor. The obligation part restricts the service requestor regarding the future usage of the released information. The Obligations in example policy (cf. Fig 8) specify that in order to access the medical record, the medical application at the client platform should be in "known good state" and the medical record can be viewed for 48 hours only.

In the extended SECTET, the XACML obligations are shipped with the protected object, and handed over to the transformation component at the client platform. It then constructs two types of policies from the shipped obli-

```
<PolicySet ....>
  <Target>
            <Subjects> PhyicianRole  </Subjects>
            <Resources>
                        MedicalSystem
            </Resources>
            <Actions>
                        getPatientRecordforWrite
            </Actions>
 </Target>                                          Server Platform Policy
   <Rule Effect="permit">
      <Condition functionId="Between">
            <Apply  FunctionId= "CurrentTime">
                        <AttributeValue> 9AM </AttributeValue>
            </Apply>
            <Apply  FunctionId= "CurrentTime">
                        <AttributeValue> 5PM </AttributeValue>
      </Condition>
   </Rule>

<Obligations>
            <Obligation ObligationId="medicalSoftwareIntegrity"  FulfillOn="Permit">
<AttributeAssignment  AttributeId="IntegrityValue" >
                        1
                        </AttributeAssignment>
            <Obligation  ObligationId="hoursAllowed"  FulfillOn="Permit">
                        <AttributeAssignment  AttributeId="hours" >
                        48
                        </AttributeAssignment>
            </Obligation>
  </Obligations>                                    Client Platform Policy

</PolicySet>
```

Figure 8. Sample XACML Policy

gations: 1) the XACML policy, which also includes the shipped obligations and; 2) the SELinux Loadable Policy Modules (LPM). Both of these policies are described in the next Section.

5. Implementation and Results

In this Section, we present the current status of the implementation within the extended SECTET. The implementation results are compiled in a way that provide a high-level view and several low-level implementation details (e.g. PKI etc) are omitted for the sake of clarity.

5.1. XACML and SELinux LPM

At the client platform, the policy decision point makes the allow/deny decisions as a last step in the usage control. An example XACML policy at the client platform looks like the same as the XACML policy shown in Figure 8 except 1) the `<Resource>` element holds the absolute path of the protected object e.g. `<Resource>/usr/medicalObject</Resource>` and; 2) the `<condition>` element contains client specific constraints. Moreover, the obligations are combined with the client XACML policy and enforced via SELinux LPM.

SELinux LPM is a way to create self-contained policy modules that are linked to existing SELinux policy without re-compiling the policy source each time. The SELinux LPM removes the key hindrance in SELinux regarding the entire policy re-compilation every time for minor changes such as adding a `user` or `role`. The LPM also supports conditional policy extensions of SELinux which enable run-time modifications to SELinux policy by associating certain parts of the policy with conditional expressions. Depending on a particular scenario, the Boolean variables within the conditional expressions can be transitioned at run-time by user space applications [25].

```
1   policy_module(medicalApp,1.0)
2   require {
3       type fs_t;
4    }
5  type medicalObject_t;
6  type medicalApp_t;
7  bool  medicalSoftwareIntegrity false;
8  bool hourAllowed false;
9  allow medicalObject_t fs_t filesystem: associate
10 allow medicalApp_t fs_t filesystem: associate
11  If (medicalSoftwareIntegrity && hourAllowed)  {
12        allow medicalApp_t medicalObject_t file:  {read getattr search}
13 } else {
14        allow medicalApp_t medicalObject_t file :  {getattr}
15 }
```

Figure 9. Sample SELinux LPM

Figure 9 shows an example SELinux conditional LPM generated from the XACML obligations. Line 5&6 declares a type `medicalObject_t` and `medicalApp_t`. Line 7&8 declares two SELinux Boolean variables. A set of permissions are defined afterwards. In particular, the first permissions allows the association of corresponding types to their actual objects in the file system (i.e. `medicalApp_t` with the medical application and `medicalObject_t` with the file containing the medical record). The actual permission that the `medicalApp_t` can read the `medicalObject_t` defined at line 12. This permissions is made conditional on the basis of Boolean variables `medicalSoftwareIntegrity` and `hoursAllowed`. These Boolean variables correspond to the obligations defined within the XACML policy (cf. Fig 8). Different daemons (services) within the client platform are responsible for transitioning these Boolean variables. In the following, a brief introduction is presented to the implemented daemons and their underlying technologies.

5.2. Integrity and Duration Verification

The *Integrity Verification Daemon* (IVD) is based on *Integrity Measurement Architecture* (IMA) – a system software for the Linux developed by IBM to provide verifiable evidence regarding the integrity of a system on which it is running. Instead of relying on the trustworthiness' of the software environment of a system, IMA builds on a hardware extension of the measured system (based on TPM). The current version of IMA requires SELinux to be disabled at boot time. However, our implementation features the IMA built on top of SELinux, a detailed account on this integration is outside the scope of this paper.

The IMA has two key components: The *Integrity Measurement* component is responsible for measuring SHA1 hash over each file and storing it in a hash table called kernel list. After taking the hash of the file, the *Integrity Protection* component stores the hash of the kernel list into a specific Platform Configuration Register (PCR) – a volatile memory built within the TPM. Beside kernel list, IMA also provides mechanism to measure and protect the integrity on the user request. This is done through the `/sys/kernel/measure` file; any file or binary listed in the `/sys/kernel/measure` will also be measured by the IMA.

In the current project state, the IVD has two main parts: 1) It writes the absolute path of the medical application to the `/sys/kernel/measure`, which is then integrity measured and protected by the IMA. 2) The second component of IVD is responsible for transitioning the corresponding SELinux Boolean variables to `true` or `false` (whenever an access request to the medical object is made) based on the integrity measured by the IMA in the kernel list. If the integrity of the medical application is in a "known good state", the `medicalSoftwareIntegrity` SELinux Boolean variable is transitioned to `true`, otherwise `false`. Once the variable is set to `false` e.g., SELinux will block all accesses to the medical object.

The duration daemon is a session oriented daemon, which keeps track of the number of hours a file is allowed based on the current system time. We assume that the system time is protected by the TPM and any illegal change in the system time will deny access to the protected medical object. The duration daemon sets the `hoursAllowed` Boolean variable based on the usage of the medical object.

5.3. Remote Attestation

The *Remote Attestation Daemon* (RAD) uses the open source *Trusted computing Software Specifications* (TSS) implementation by IBM called TrouSerS [37]. The TrouSerS project aims at implementing TSS specification and providing an API that enable an application's use of a platform TPM. That is to access the TPM chip and its configuration registers from the user space applications. For example, the integrity of the medical application is remotely attested to the data provider (the hospital) in the following steps:
1) The Remote Attestation Daemon creates a public-private signing key pair by calling the function `Tspi_key_createKey()`. The key is attested by the storage root key (SRK) which is burned into the hardware chip – TPM. 2) After creating the key, the

Daemon, queries the specific Platform Configuration Register (PCR) for the hash of the kernel list using the function `Tspi_composite_getPCRValue()`. 3) The hash is the signed using the `Tspi_hash_sign` function which takes (as input) the hash and the public part of the signing key. 4) The signature and the public key are stored in separate files and sent to the data provider over a web service.

A *Properties Verification Daemon* (PVD) at the data provider site then uses OpenSSL to verify and attest the properties sent by the client platform. Assuming a Certification Authority (CA) have verified the pubic key sent by the data consumer, the Properties Verification Daemon uses the `RSA_Public_Decrypt()` to first decrypt the signature with the public key, extract the medical application hash from the kernel list and then compare it with the original hash of the medical application stored with the data provider (using `memcpy()` function).

We came to the conclusion that the combination of SELinux and XACML policies provide a generic solution, as it can express fairly complex TC based secure information sharing scenarios. Currently, we are extending our framework to include automatic transformations from the high-level SECTET-PL specifications to SELinux policies.

6. Related Work

Sandhu et al [33, 34] described a model framework called PEI for secure information sharing with TC technologies, which is within the same problem space as our work. The framework consists of three model layers: policy, enforcement, and implementation. Our work presented in this paper can be regarded as within the policy and enforcement models in PEI. Particularly, we develop extended SECTET for high level policy specifications through UML and predicate languages, and we leverage TC and SELinux for enforcement mechanisms. However, our work has some notable difference from PEI. First of all, instead of focusing in small-scale and group-based object distribution environment in PEI, our work targets on highly distributed web services architecture such as a healthcare network. Secondly, in PEI, the policies are typically formal or semi-formal, while the policy modelling in our work is more engineering-oriented, and we are developing automated tools to transform high level policies to SELinux polices that can be enforced in verified platforms. Thirdly and most importantly, in PEI-based information sharing, TC requirements are enforced with high-level attestation protocols. However, our work integrates TC-related requirements with SELinux such that the policy model of SELinux is fundamentally extended, thus we can leverage the fine-grained access control of SELinux to dynamically enforce TC requirements.

Remote Attestation for web services called *WS-Attestation* has been proposed in [38] and with a slight variation in [36]. However, both approaches directly bind the web services standards to the TCG technologies to increase trust and confidence in integrity reporting. In our case, shipping extended SELinux policies with the protected object/information are better suited for secure information sharing and leverages policy oriented TC approach which is a novel aspect.

A distributed usage control policy language and its enforcement requirements are presented in [24, 6]. Similar to our objective, this work targets on control over data after its release to third parties. However, the significant difference between this and our work is that our work relies on the underlying TC services of a platform.

Jaeger et al [16] have recently proposed a framework called PRIMA which is an extension to IMA [32]. Their framework is basically concerned with the improvement of efficiency during the integrity measurement of the platform by limiting the number of measured entities. However, compared to our approach, PRIMA does not discuss the specification of TC-related requirements by any means. Their proposed model can be incorporated within the extended SECTET for efficient integrity measurement.

7 Conclusion & Evaluation & Outlook

In this paper, we present SECTET – a model-driven framework for rendering TC-related requirements at a higher level of abstraction. To the best of our knowledge, this is the first approach for the integration of TC-related requirements into distributed application development with an emphasis on modelling approach. We also demonstrate the use of SECTET-PL – a predicative language for the specification of TC dependent constraints (or obligations). SECTET-PL predicates are then transformed to equivalent XACML and SELinux policies.

Our performance evaluation shows that there are two most expensive operations 1) fingerprinting the file for the first time and maintaining it in the kernel list (5765 μs) and 2) the signature generation which is attested by the TPM SRK (2141927 μs). The reason is obvious: TPM is a slow chip and depending on the size of the files, it can take even longer to perform hardware related steps.

Currently, we are extending the implementation framework to tie the remote attestation with the decision process at the client platform. In order to provide, automatic transformations, we are also developing an eclipse plug-in based tool support for QVT.

References

[1] Experiences Implementing a Higher-Level Policy Language for SELinux in Second Annual SELinux Sym-

posium, 2006. Baltimore, Maryland. http://selinux-symposium.org/2006/abstracts.php.

[2] Security-Enhanced Linux (SELinux). www.nsa.gov/selinux/.

[3] The Platform for Privacy Preferences 1.0 (P3P1.0) Specification. W3C Recommendation 16 April 2002.

[4] Trusted computing group (tcg). https://www.trustedcomputinggroup.org/specs/.

[5] Uml profile for corba components, may 2003. OMG Document mars/03-05-09.

[6] A. Pretschner et al. Distributed usage control. *Communications of the ACM*, 49(9):39–44, September 2006.

[7] M. Alam, R. Breu, and M. Hafner. Model-Driven Security Engineering for Trust Management in SECTET. JOURNAL OF SOFTWARE, VOL. 2, NO. 1, FEBRUARY 2007, ACADEMY PUBLISHER.

[8] M. Alam, M. Hafner, and R. Breu. Constraint based Role Based Access Control for modelling administrative constraints in the SECTET. In *Proceedings of the ACM PST 2006 - International Conference on Privacy, Security and Trust, October 30th, 2006 - November 1st, 2006.*

[9] M. Alam, M. Hafner, and R. Breu. Modeling Authorization in a SOA based Distributed Workflow. In *IASTED Software Engineering 2006, isbn 0-88986-572-8.*

[10] http://www.antlr.org.

[11] R. Breu and G. Popp. Actor-centric modelling of access rights. In *FASE 2004*. Springer LNCS Vol. 2984, p. 165-179, 2004.

[12] K. Czarnecki and S. Helsen. Administration Model for Or-BAC. In *Classification of Model Transformation Approaches*, volume 19, OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture.

[13] A. M. Devices. *AMD Secure Virtual Machine Architecture Reference Manual*. AMD, 2005.

[14] D. Grawrock. *The Intel Safer Computing Initiative Building Blocks for Trusted Computing*. Intel Press, http://www.intel.com/intelpress/sum_secc.htm, 2005.

[15] V. Haldar, D. Chandra, and M. Franz. Semantic remote attestation - a virtual machine directed approach to trusted computing.

[16] T. Jaeger, R. Sailer, and U. Shankar. PRIMA: policy-reduced integrity measurement architecture. In *SACMAT '06: Proceedings of the eleventh ACM symposium on Access control models and technologies*, pages 19–28, New York, NY, USA, 2006. ACM Press.

[17] T. Jaeger, R. Sailer, and X. Zhang. Resolving constraint conflicts. In *SACMAT '04: Proceedings of the ninth ACM symposium on Access control models and technologies*, pages 105–114, New York, NY, USA, 2004. ACM Press.

[18] M. Alam. Model-Driven Security Engineering for the realization of Dynamic Security Requirements in Collaborative Systems . Appeared in PhD Symposium of IEEE/ACM Models 2006 LNCS 4364.

[19] M. Alam et al. Model Driven Security for Web Services (MDS4WS). In *INMIC 2004,Digi Obj Id 10.1109/INMIC.2004.1492930.*

[20] M. Alam et al. Modeling Permissions in a (U/X)ML World. In *IEEE ARES*, 2006.

[21] M. Hafner et al. Modeling Inter-organizational Workflow Security in a Peer-to-Peer Environment. In *IEEE ICWS 2005,ISBN: 0-7695-2409-5.*

[22] M. Hafner, M. Alam, R. Breu. A MOF/QVT-based Domain Architecture for Model Driven Security . In *IEEE/ACM Models 2006 LNCS 4199.*

[23] M. Hafner., R. Breu, and M. Breu. A Security Architecture For Inter-organizational Workflows-Putting WS Security Standards Together. In *ICEIS 2005,ISBN: 972-8865-19-8.*

[24] M. Hilty et al. Usage control requirements in mobile and ubiquitous computing applications. In *Proc. of Intl. Conf. on Systems and Networks Communications*, 2006.

[25] F. Mayer, K. MacMillan, and D. Caplan. *SELinux by example: using Security Enhanced Linux.* 2006.

[26] UML 2.0 OCL Specification. http://www.omg.org/docs/ptc/03-10-14.pdf.

[27] S. Pearson. *Trusted Computing Platforms: TCPA Technology in Context*. Prentice Hall PTR, Upper Saddle River, NJ, USA, 2002.

[28] R. Breu et al. Model Driven Security for Inter-Organizational Workflows in e-Government. In *TCGOV 2005,Proceedings. ISBN 3-540-25016-6.*

[29] R. Breu et al. Web service engineering - advancing a new software engineering discipline. In *ICWE 2005, LNCS 3579 Springer 2005.*

[30] Role Based Access Control avialable at. csrc.nist.gov/rbac/.

[31] R. Sailer, T. Jaeger, X. Zhang, and L. van Doorn. Attestation-based policy enforcement for remote access. In *CCS '04: Proceedings of the 11th ACM conference on Computer and communications security*, pages 308–317, New York, NY, USA, 2004. ACM Press.

[32] R. Sailer, X. Zhang, T. Jaeger, and L. van Doorn. Design and implementation of a tcg-based integrity measurement architecture. In *USENIX Security Symp.*, 2004.

[33] R. Sandhu, K. Ranganathan, and X. Zhang. Secure information sharing enabled by trusted computing and pei models. In *ASIACCS '06: Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pages 2–12, New York, NY, USA, 2006. ACM Press.

[34] R. Sandhu, X. Zhang, K. Ranganathan, and M. J. Covington. Client-side access control enforcement using trusted computing and pei models. In *Journal of High Speed Network, Special issue on Managing Security Polices: Modeling, Verification and Configuration*, pages 15(3): 229 – 245, 2006.

[35] Shiboleth protocols and profiles: August 2005. http://shibboleth.internet2.edu/shib-intro.html.

[36] Z. Song, S. Lee, and R. Masuoka. Trusted web service. In *The Second Workshop on Advances in Trusted Computing (WATC '06 Fall)*, 2006.

[37] Trusted Computing Software Specification implementation, available at. http://trousers.sourceforge.net/.

[38] S. Yoshihama, T. Ebringer, M. Nakamura, S. Munetoh, and H. Maruyama. Ws-attestation: Efficient and fine-grained remote attestation on web services. In *ICWS '05: Proceedings of the IEEE International Conference on Web Services (ICWS'05)*, pages 743–750, Washington, DC, USA, 2005. IEEE Computer Society.