Journal of Universal Computer Science, vol. 15, no. 15 (2009), 2916-2936
submitted: 1/2/09, accepted: 29/8/09, appeared: 1/9/09 © J.UCS

SeAAS - A Reference Architecture for Security Services in SOA

Michael Hafner
(University of Innsbruck, Innsbruck, Austria
m.hafner@uibk.ac.at)

Mukhtiar Memon
(University of Innsbruck, Innsbruck, Austria
mukhtiar.memon@uibk.ac.at)

Ruth Breu
(University of Innsbruck, Innsbruck, Austria
ruth.breu@uibk.ac.at)

Abstract: Decentralized security models and distributed infrastructures of scenarios based on Service Oriented Architectures make the enforcement of security policies a key challenge – all the more so for business processes spanning over multiple enterprises. The current practice to implement security functionality exclusively at the endpoint places a significant processing burden on the endpoint, renders maintenance and management of the distributed security infrastructures cumbersome, and impedes interoperability with external service requesters. To meet these challenges, we propose a reference security architecture that transposes the model of Software as a Service to the security domain and thereby realizes Security as a Service (SeAAS). The proposed architecture goes beyond the mere bundling of security functionality within one security domain. We illustrate the concepts of SeAAS at work with the requirement of fair non-repudiation. The architecture complements the SECTET framework for model-driven security engineering. [1]

Key Words: Security as a Service, Service Oriented Architecture, Security Requirements
Category: D.2.10, D.2.11

1 Introduction

Inter-organizational workflows spanning multiple domains of business partners involve the sharing of sensitive resources. The examples are numerous and ever-growing. In healthcare, a patient's electronic healthrecord stored with a hospital may be updated with a radiography produced by an external specialist, complemented with a diagnosis together with a regular update of the medication prescribed by the patient's practitioner. Or a company's financial statement may be forwarded to auditors before being turned in as an electronic tax declaration with financial authorities (e.g., [Hafner and Breu 2008, Hafner et al. 2006]). Those large-scale software systems can be characterized as heterogenous, distributed systems spanning across many enterprises under the control of as many "ownership domains". Often, they are realized based on the blueprint of Service Oriented Architecture (SOA). However, the decentralized security models and dis-

[1] This work was partially supported by the SecureChange (ICT-FET-231101) EU project and the SECTISSIMO (P-20388) FWF project.

tributed infrastructures of SOA turn the enforcement of security requirements into a major challenge.

Technical interoperability was addressed first and with some success. To make sure companies were using "compatible" technology for cooperation with their peers, software engineers and architects could turn to the paradigm of SOA with its standardized technical underpinning, the stack of Web services standards and technologies. However, up until now, the standards only address basic security requirements (the triad of traditional information security, namely confidentiality, integrity and availability) and resolve issues at a low, technical level. This makes security engineering incredibly complex and – as a consequence – implementations error-prone.

According to current practice, security infrastructures enforce security exclusively at the servcie endpoint. They ignore the pecularities of SOA's decentralized peer-to-peer architecture, which outmodes traditional security solutions and mechanisms, among them the concept of perimeter security and centralized security models [Peterson 2005, Peterson 2009]. Besides placing a significant processing burden on service nodes, endpoint security renders maintenance and management of the distributed security infrastructures cumbersome, and impedes interoperability with external service providers and requesters. To meet these challenges, we propose a reference security architecture that transposes the model of Software as a Service to the security domain and thereby realizes Security as a Service (SeAAS). The proposed architecture goes beyond the mere bundling of security functionality within one security domain as it realizes complex security requirements for processes invovling two or more domains. The solution complements the SECTET framework for model-driven security engineering [Hafner and Breu 2008].

We structured our work as follows. Section 2 introduces a contemporary example from the healthcare industry. In Section 3, we develop our line of argument motivating the need for Security as a Service as a paradigm for SOA security and present related work. We introduce an architectural blueprint for an infrastructure leveraging security services in Section 4. In Section 5, we show how the proposed solution solves complex security requirements illustrated with an example of fair non-repudiation. In Section 6 we have explained various Web services based security standards and technologies, which we use in the Reference Architecture. We close our paper with a conclusion, sided by a discussion of the challenges ahead and future work in Section 7.

2 Motivating Example

Our scenario draws the security requirements from use cases of the healthcare industry. They were elaborated in the context of national initiatives in Europe with the aim to realize the Electronic Health Record (EHR) [Becker and Sewell 2004, Alam et al. 2007, Hafner and Breu 2008]. We begin with a functional description in Section 2.1 and proceed to security requirements in Section 2.2.

2.1 The Electronic Health Record - A Use Case

Figure 1 shows the various stakeholders modeled as roles and their interactions with an EHR system modeled as message exchanges in a typical scenario. Security-relevant communication is indicated in red.

Markus Maier (role `Patient`) goes to see Dr. David Daum, his family doctor (role `General Practitioner`) for his yearly medical check-up. In step 1, Dr. Daum accesses Markus Maier's **Electronic Health Record** over the centralized EHR service (ELGA[2]). In its essence, an EHR represents a consolidated virtual medical record assembled from information distributed across various healthcare providers, which produced clinical information during past consultations and treatments. In Markus Maier's case these were the *City Hospital* and the *City Sanatorium* as `Public-` and `Private Healthcare Provider`, respectively & *City Health Insurance* as a `3rd Party Institution`. After a first examination, Dr. Daum decides to refer Markus Maier to the radiologist Dr. Rudolf (role `Specialist`). He does so by issuing an electronic **Referral** which updates the EHR (step 3). In consultation with his patient, Dr. Rudolf accesses Markus Maier's EHR (step 4), and updates his EHR with the produced **Radiography** (step 5). Afterwards, Markus Maier may have to submit to a couple of further checks (not shown here), e.g., have blood samples checked by a medical laboratory, submit to a stress electrocardiogram with an internal specialist etc., before he pays his final visit to Dr. Daum for a discussion of the medical statement. On this occasion, Dr. Daum updates Markus Maier's EHR with the **Medical Statement**.

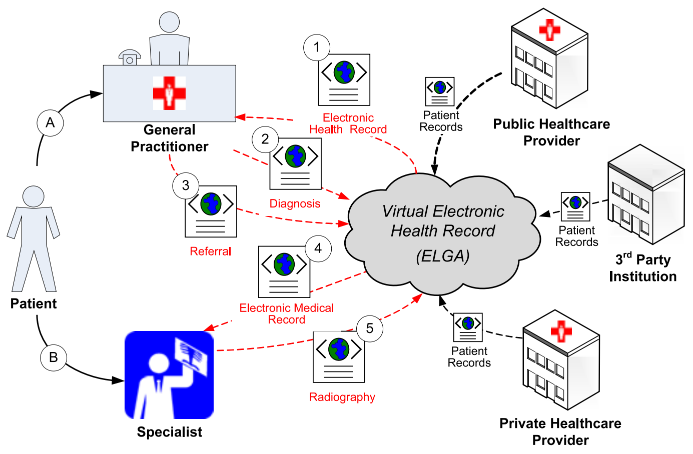

Figure 1: Roles and Message Exchanges in a Distributed Healthcare Scenario.

[2] ELGA stands for "Elektronische Gesundheitsakte", the German acronym for EHR [Arge ELGA].

2.2 Security Requirements

Based on that common scenario, we can identify a broad array of security requirements. Sections 4, 5 and 6 discuss how our architecture realizes these requirements. Details are illustrated taking Non-repudiation as an example for a complex security requirement.

Authentication (And Identity Management). The EHR infrastructure facilitates the identification, registration and authentication of professional users – (be they humans or services) – based on digital certificates and public key technology. Users access medical information in the EHR based on role credentials issued by ELGA Certification Authorities. The credentials are valid across the security domains of involved stakeholders in the overall scenario. Nevertheless, a single health organization is very likely to manage identities within its own security domain running a "local" certification or registration authority. The local identities of users and applications which interoperate with the EHR system are mapped to "global" identities managed by the ELGA Certification Authorities. For example, although Dr. Daum may have authenticated himself to his local application, he would have to authenticate himself a second time with his global credentials to the EHR system to access his patient's records via ELGA.

Authorization. Role credentials define the permissions to access health records. Realizing the principle of Least Privilege, users are given those privileges necessary to perform their job as specified by system roles (e.g., Specialist, Healthcare Provider etc.). This entails the necessity for a fine grained protection of the resource. For example, the role `Pharmacist` only needs access to those parts of the EHR containing the prescription of medication. In the EHR System, there is a default Permission-Role assignment that may be overwritten by the record owner[3]. Although a user holding the role `General Practitioner` may be given the most comprehensive access to his patient's medical records (if he is the primary care physician), a `Patient` may confine his privileges. Markus Maier may not want his father-in-law working as a psychiatrist in the City Hospital to see medical records about a psychotherapeutic treatment he had to undergo a couple of years ago due to a mental problem. So he could define a *Negative Access Permission* to these records for his father-in-law. Other complex authorization policies that come into play in standard use cases are the *Delegation of Rights*, *Four-Eyes-Access-Control*, *Break-Glass-Policy*, and *Dynamic Access Control*. A comprehensive treatment on the modeling and enforcement of complex access control policies in healthcare with the SECTET framework, can be found in [Alam et al. 2007] and [Hafner and Breu 2008].

Non-repudiation. This requirement aims at preventing parties in a communication from falsely denying having taken part in that communication. In our context, enforcement of Non-repudiation is typically transparent to users [Agreiter et al. 2008]. It comes in two flavors. *Non-repudiation of Reception* requires the addressee to return a proof of receipt (e.g., a signed message carrying a time-stamp) to the sender to be kept in case

[3] As for now the discussion about the actual ownership of records is still not resolved, see [Donner 2004]

dispute resolution is needed. In our scenario, Dr. Daum and Dr. Rudolf will both get a proof of receipt from the EHR system after having updated Markus Maier's EHR with the produced documents and artefacts (Referral, Radiography, and Medical Statement). Complementarily, *Non-repudiation of Origin* requires the sender to produce a proof of submission and make it accessible to the receiver. The EHR system will log the updates to Markus Maier's EHR. The security infrastructure takes care of producing and consuming the messages and initiating logging activities.

Security Compliance and Governance. Security compliance aims at the detection of deviation from allowed behaviour, specified interaction patterns, or message structures. In the current state of the SECTET framework, we view security compliance in its narrowest sense. It defines the adherence of messages to predefined structures or interaction patterns based on supported security infrastructures and mechanisms (e.g., type of tokens, encryption, signature algorithms, request-reply, one way etc.). It is enforced by the security infrastructure and is offered as a service to local applications and users.

3 Security as a Service - Making the Case

In this section, we motivate the need for *Security as a Service* (SeAAS) as a paradigm for security architectures and present related work as we pursue our line of argument.

3.1 Limitations of Endpoint Security

According to current best practice, Web service security is mostly enforced at the physical node providing the service or the component proxying the service (henceforth, we will call the service and / or its proxy component *service endpoint*). In a typical Web services based request, the service endpoint applies basic cryptographic processing to inbound and outbound messages leveraging XML based standards [Imamura 2002, Bartel et al. 2002, Atkinson et al. 2002]. It extracts and validates tokens of incoming messages, decrypts encrypted parts, validates signatures etc. Outbound messages are processed and their structure extended so to comply with policy requirements as imposed by the service endpoint's communication peer.

Traditional endpoint security falls short on two fundamental issues of large-scale business solutions. The first issue is related to the complexity of security engineering. Web services standards and technologies are constantly evolving. New security standards are added to the stack of Web services standards to cover new requirements and use cases (cf [InnoQ 2006] for an overview). This fast moving target is a challenge to security experts and software engineers alike. Traditional methods of software engineering can hardly cope with the plethora of standards combined with the complexity of security solutions needed for the realization of enterprise-wide and inter-organizational business solutions. This is often considered to be a major obstacle to the rapid adoption of Web services as a reference platform to large-scale solutions. Another issue is related to the consistent enforcement of security policies in enterprise-level solutions.

These environments are characterized as large-scale distributed architectures with thousands of services deployed on hundreds of endpoints and possibly as many internal and external consumers. Nevertheless, any access decision has to be attributable to security policies that meet the obligations imposed by laws and regulations like the Sarbanes-Oxley Act and complying corporate governance policies. This necessitates policy management concepts and security mechanisms that guarantee consistent enforcement of security policies in distributed, heterogenous environments.

3.2 Declarative Security

The concept of declarative security was a first step to cope with the issues exposed in the preceeding section. It addresses three challenges. *1. Development.* Security concerns are separated out of actual application and service development. The burden of security enforcement is shifted from service developers and service requesters to security experts who codify security requirements into policies based on rules. This eases the realization of security-critical use cases. XACML is an example standard for declarative access control [OASIS 2006]. It proposes a declarative access control policy language implemented in XML and an architectural blueprint for the communication between infrastructural components. *2. Interoperability.* Security requirements on message structure and syntax are codified as rules in the machine readable XML standard WS-Security Policy [Bajaj 2006] and advertised to potential service requesters. This improves interoperability of security solutions that cross organizational boundaries. *3. Policy Management.* Security policies expressed in declarative statements can be checked for consistency and – once consolidated – distributed to the application which is meant to enforce them. This fosters the consistent application of policies across all solutions in an enterprise.

Even with declarative security, the realization of security-critical inter-organizational scenarios still faces two major hurdles. For one, with enforcement left to the endpoint, security solutions are scattered over the service landscape. This means that in order to keep up to date with evolving technologies and changing security requirements that demand new functionality, security engineers have to propagate the changes to every single endpoint – a very inefficient form of reusability. Secondly, the state of the art in Web service and SOA security technology indicates that for the moment only very basic security requirements like those based on the application of cryptographic operations are realized. This is actually the main criticism advanced by industries that are primarily concerned about complex security requirements in an inter-organizational setting like in healthcare and e-government. Here, use cases accommodate security requirements derived from complex industry regulations and laws (cf Section 2.2).

Existing standards and specifications do not address these security requirements at all. The reason is, that their realization would overstrain the capacities of a single endpoint, either in terms of the complexity of the underlying security concept (e.g., the protocols of non-repudiation), or in terms of the processing power (e.g., evaluation of log-files for security monitoring), or in terms of functionality (e.g., basically stateless service endpoints are not supposed to have all information necessary to infer unusual user behaviour for fraud detection). A very technical account on how to realize declarative security that covers basic security for Web services (authentication, confidentiality, integrity) in an SOA is given in [Kanneganti and Chodavarapu 2007].

3.3 The Enterprise Service Bus

These practical problems (processing burden, complexity of security) and conceptual issues (statelessnes of services) suggest the outsourcing of security tasks to an architectural component with the needed capabilities. A very promising approach is put forward by the paradigm of SOA: the *Enterprise Service Bus* is the technical backbone of a SOA landscape. This centralized communication infrastructure is responsible to provide interoperability between heterogenous systems. This means connecting them in a loosely coupled way (independent of technical protocol details), mapping data-types, transforming formats and guaranteeing transparent routing dealing with technical aspects, such as load balancing and failover. It is considered to be the ideal candidate to offer value-added services such as security, monitoring and debugging [Josuttis 2007].

Up until now, security has only been integrated at a very basic level. For example, [Kanneganti and Chodavarapu 2007] gives a detailed technical account on how to secure SOAs, but only covers authentication, authorization, confidentiality and integrity. The focus is set on a centralized setting confined to a single security domain (as opposed to the decentralized setting of inter-organizational scenarios presented in the next section). [Hafner et al. 2006, Hafner and Breu 2008] give a detailed account on issues related to the realization of security-critical decentralized SOA.

[Rademakers and Dirksen 2008] describes how to realize the concept of a centralized communication infrastructure – the ESB – with open source software in all details. Security is only covered at a very basic level. In [Hinton et al. 2005] the authors move a step further and discuss security as an infrastructure service in the context of an Enterprise Service Bus (ESB) and other patterns for the deployment of an an SOA-security infrastructure. Nevertheless their solution only covers the standards basic security services (e.g., authentication, key management etc.).

3.4 The SECTET Framework for Model Driven Security

Model Driven Security is an engineering paradigm that specializes Model Driven Software Engineering towards information security. It pursues two objectives: first, the integration of security aspects at an early stage of the engineering process and second, to shift the burden of security implementation from the software engineer to the security engineer. The term Model Driven Security was coined in [Basin et al. 2006]. The paper describes a software development process that supports the integration of access control requirements into system models. The models form the input for the generation of .net and J2EE security infrastructures. [Juerjens 2004] presents a framework for the formal verification of basic security requirements for security protocols based on UML models. Focusing on Service Oriented Architecures, in [Satoh et al. 2006] the authors propose a framework for the platform-independent configuration of security infrastructure with authentication information.

Pursuing a much broader goal, the SECTET [Hafner and Breu 2008] framework supports business partners during the development and distributed management of decentralized peer-to-peer scenarios. Primarily developed for the realization of decentralized, security critical collaboration across domain boundaries – so-called inter-organizational workflows, it realizes a domain architecture aiming at the correct technical implementation of domain-level security requirements. It consists of three core components:

1. **Security Modeling**. The modeling component supports the collaborative specification of a scenario at the abstract level in a platform independent context. The component implements an intuitive domain specific language, which is rendered in a visual language based on UML2 for various modeling tools. The modeling occurs at a level of abstraction appropriate to bridge the gap between domain experts on one side and engineers on the other side, roles chiefly involved in two different phases of the engineering process – the requirements engineering and the design phase respectively.

2. **Code Generation & Model Transformation**. Model information is translated it into platform independent models (PIM) based on security patterns and protocols enforcing security requirements. The PIMs are refined into platform specific models of various granularity until they can be mapped into configuration code for the components of the target architecture. The layered approach is detailed in [Memon et al. 2008].

3. **Web services Based Reference Architecture**. The architecure specifies a Web services based target runtime environment for local executable workflows and back-end services at the partner node. The workflow and security components implement a set of workflow and security technologies based on XML- and Web services technology.

The SECTETreference architecture as presented in [Hafner and Breu 2008] enforces security mostly at the service endpoint. As already exposed, the approach exhibits significant limitations, especially when it comes to the realization of the complex security requirements. In the present contribution we propose an alternative blueprint realizing *Security as a Service*.

3.5 Security as a Service

Security as a Service was introduced a couples of years ago in a series of publications. [Peterson 2005] first advocated the transition from traditional perimeter security to the concept of Service Oriented Security – a framework for risk analysis and management focusing on assets of a decentralized (service based) software architecture. Security is realized through decoupled, composeable services. The contribution focuses on identitiy and risk management, omitting complex security requirements. [Peterson 2009] discusses how the field information security failed to resolve key challenges of SOA security and arguments very much inline with our strategy that workable solutions have to move beyond traditional information security which only considers the CIA traid. Other publications on various aspects of security services in SOA that we discuss in this paper are [Hinton et al. 2005, Lopez et al. 2005, ORACLE 2008].

We define Security as a Service (SeAAS) as the delivery of security functionality over infrastructure components in a service-oriented manner. For SOA, this means that security services are accessed through common Web services technologies and standards. Our definition thus goes beyond the common understanding which confines SeAAS to the practice of delivering traditional security application functionality (e.g., anti-virus software, anti-spyware, etc.) on-demand over the Internet (e.g., [McAfee 2008, Microsoft 2006, Symantec 2006]). We identified the following security services (in order of increasing complexity):

1. Cryptographic and message processing services ensure basic confidentiality and integrity e.g., en-/ decryption of XML documents, signature validation etc.

2. Security inter-operability services facilitate interoperability of security mechanisms with external partners e.g., mapping of a user credentials to a kerberos token. Services can be provisioned by an internal or an external security token service.

3. Authentication is a basic service necessary to all other requirements. Local or external service requesters are identified and authenticated reyling on local identity stores and / or external identity providers.

4. Authorization services provide access control to ressources. Authorization policies can be very complex. We cover static and dynamic role-based access control, four-eyes principle, negative access permissions, delegation, and break-glass policies.

5. Security compliance services check inbound messages in inter-domain communication for compliance with stated security requirements, e.g., valid and complete messages, presence of tokens, format, etc.

6. Protocol based security services are statefull services executed between two or more partners. A very prominent example is non-repudiation of sending or receiving in inter-domain communication.

7. Security monitoring & auditing services facilitate business- or application level security requirements, e.g., fraud and intrusion detection.

To cope with all classes, the proposed architecture goes beyond the mere bundling of security functionality within one security domain. For example, the execution of protocol based security (e.g., non-repudiation) realizes complex security requirements for processes involving two or more domains.

It is noteworthy, that in some cases, especially for the sake of security interoperability and efficient manageability, endpoints already rely on centralized services. We identified the following cases. *1. Authentication*. To authenticate a service requester, an endpoint commonly relies on a centralized identity store for identity management in its own security domain. In [Lopez et al. 2005], the author identifies dedicated security services for advanced authentication and authorization requirements. *2.Interoperabil-*

ity. In some cases, an endpoint may issue a request for a token mapping with a trusted 3rd party security token service. Here, interoperability seems to make the case for a "service-ization" of security. It is a way to cope with the heterogeneity of distributed, inter-organizational processes with different infrastructure owners organized into separated security domains. *3. Authorization*. Within a security domain, authorization is enforced at the endpoint, but relies on a central policy decision point for decisions on access requests. The XACML dataflow model [OASIS 2006] defines a reference architecture for the enforcement of access control in a service based environment whithin a security domain.

4 Architectural Blueprint for Security As A Service

4.1 SeAAS Architecture

Figure 2 shows the conceptual architecture for the proposed SeAAS approach (EI Pattern names in italics). The upper part shows the *ELGA Healthcare Services Architecture*. *Service Endpoints* provide business functionality. ELGA offers a number of healthcare services, such as access/update a patient's EHR, add Radiography to EHR etc. The service endpoints are decoupled from the security and messaging components. Inbound and outbound messages are delivered over the ESB. A business message contains service requests and responses whereas a security message contains security protocol data. An ESB handles internal communication among the various components of a domain and external communication with business and security components of other domains. It intercepts inbound requests and forwards them to the *SeAAS Engine* for security evaluation. The SeAAS Engine is the central part of the SeAAS Component. To evaluate security, it retrieves the applicable security policy from *Policy Repository*. The security policy defines the security requirements for a particular request. The SeAAS Engine parses the policy, retrieves the security requirements and decides which security services will be needed to fulfill those requirements. It composes a security process to call those services in an appropriate order. For simplicity, our prototype currently uses a static process with a pre-defined order of execution of security services. For example, Authentication, Authorization and Non-repudiation Services are executed in following order: *Non-repudiation* ⇒ *Authentication* ⇒ *Authorization*.

The SeAAS Component offers security functionality as a set of *Security Services* implemented as *Security Components*. *Primitive Security Services* consist of Encryption, Signature and Time-stamping services. All other services (e.g., Authentication, Authorization, etc.) are considered *Advanced Services*. They leverage primitive security services. For example, the Non-repudiation service uses them to encrypt, sign and timestamp evidence. Primitive security services need keys/certificates, which are stored in the *Key Repository*. One key feature of the SeAAS architecture is the realization of security through decoupled components so to attain technology and language independence. All components can be implemented in any language and/or technology without

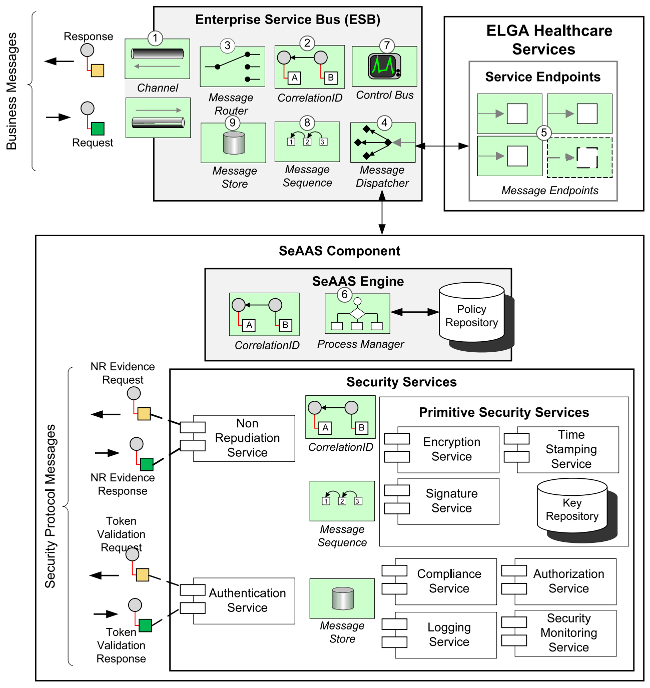

Figure 2: SeAAS Conceptual Architecture.

any inter-dependence. Communication is fully message-oriented and is carried out over an ESB. Our prototype is based on Apache ServiceMix [ServiceMix 2008] – an open source ESB.

4.2 SeAAS Component

Deployed within a security domain, the *SeAAS Component* consists of a number of security services. Depending upon the requirements of the domain, new security services can be deployed during runtime. The *Policy Repository* (PR) and the *PKI Repository* offer supporitng services. The PR holds the policies which specify security requirements, whereas PKI Repository is a local store for keys and certificates.

1. Authentication. The authentication Service provides intra- and inter-domain authentication. In case of an internal request, the authentication service validates the user's local identity and sends the signed authentication decision to the endpoint. For a request from an outside domain, the authentication service first resolves the identity of the external user: it contacts the external identity provider (e.g., a *Security Token Service* (STS)) using *WS Interface*. After the STS validates the user, the authentication service creates a security context. This provides the functionality for *Identity Federation*.

2. Authorization. The Authorization Service verifies permissions assigned to users.

They are defined in the policies stored in *Policy Repository*. Based on the policy, the service takes a decision and sends the result to the service endpoint for enforcement.

3. Non-repudiation. This service executes an out-of-band non-repudiation protocol between requester and the endpoint and stores evidence for dispute resolution (Section 5 is dedicated to a detailed discussion of non-repudiation).

4. Security Compliance. This service verifies the compliance of an inbound message with the security policy of the target service endpoint. The security policy of service endpoint defines the supported security mechanisms such as types of tokens, encryption and signature algorithms, message parts to be protected etc. The authentication service depends upon the evaluation performed by the compliance service. If a request is compliant, then the authentication service proceeds with token validation.

5. Security Monitoring. This service monitors significant security events generated by the security services of the SeAAS Component. For instance, the compliance service reports a security event, if a message does not meet an endpoint's security policy. The Non-repudiation service notifies a protocol failure, when the external endpoint does not follow the Non-repudiation protocol. The monitoring service of a domain's SeAAS component forwards these events to a central service accessible to all domains. The purpose of monitoring security centrally is to receive the security events from different domains and notify responsible and affected endpoints.

6. Logging. This service logs notifications sent by endpoints related to various business requests, responses, errors and exceptions.

Externalising security functionality as a set of services significantly reduces endpoint complexity. Moreover, the composition of security services as SeAAS components facilitates the deployment and the configuration of exisitng and new security components at deployment time and even during runtime.

4.3 Enterprise Integration Patterns

Patterns provide sound solutions to commonly known problems. In today's business world application integration is more complex, as the systems are loosely-coupled and use heterogeneous technologies. Message-oriented integration (MOI) aims at achieving integration among heterogeneous applications based on EI patterns. Figure 2 shows the EI patterns used by components instead of language-specific modules to realize message-oriented communication. EI patterns will help the security developer implement proposed SeAAS architecture, irrespective of what tools, technologies and languages she is using. A full catalogue of Enterprise Integration patterns is presented in [Hohpe and Woolf 2004]. As dsicussed nelow, we used some of the ose patterns, which are appropriate for designing the SeAAS components (EI patterns are circle-numbered in Figure 2 as well as in the text below).

The ESB uses *Channels (1)* to send/receive business and security protocol messages. As the integration of components in SeAAS is message-oriented, there should be certain mechanisms to relate the incoming and outgoing messages at any component. The *CorrelationID (2)* is used for matching requests and responses by the business and security components. Every message that enters and leaves the boundaries of a domain or a component in the domain is assigned a unique CorrelationID. The global correlationID for a domain is assigned by the ESB, whereas the local correlationIDs are assigned by components such as SeAAS Engine and Security Services. The *Message Router (3)* pattern is used for routing, so that the ESB sends the messages to appropriate destinations. A *Message Dispatcher (4)* consumes messages from Message Router and distributes them to their destinations. ESB uses this pattern to dispatch (business/security) messages to SeAAS Engine, Security Services, Service Endpoints and external domains. The service endpoints use *Message Endpoints (5)* pattern to indicate a client of messaging system i.e. ESB to send/receive messages. The *Process Manager (6)* pattern is used to model the SeAAS Engine for security process composition. The *Control Bus (7)* pattern indicates that the ESB sends logging and security events to the Logging and Security Monitoring components, which monitor failures, exceptions and security violations. The order of the messages is important, when the security services send and receive security protocol messages. The *Message Sequence (8)* pattern is used by security services to maintain the required order of security protocol messages. The ESB uses a separate Channel to store a copy of the messages into *Message Store (9)* to analyze the message before it delivers it to the target destination. The non-repudiation service uses this pattern to store the signed messages in a local persistent database. Similarly, Logging and Security Monitoring services store event notifications associated to certain message for security analysis.

5 Realizing Complex security Requirements with SeAAS

Complex security requirements are realized through advanced security services. Here, we illustrate the working of one of those services taking *Non-repudiation* as an example.

There is much research related to the non-repudiation protocols [Zhou et al. 1999, Markowitch et al. 1999, Kremer et al. 2002]. Most is focused to the achievement desired properties like *Fairness* and *Timeliness*. Another issue extensively covered in research is concerned with the design of protocols with or without a Trusted Third Party (TTP) [Markowitch et al. 2001]. The protocols achieve non-repudiation among two or more protocol participants. A non-repudiation protocol is a cryptographic protocol that provides irrefutable evidence to its participants. A protocol is called *Fair*, if it provides the originator and the recipient of the message, with some evidence after completion of the protocol, without giving a participant an advantage over the other at any stage. Our design of NR protocol is based on ZG's protocol to achieve non-repudiation properties of fairness and timeliness [Zhou et al. 1999].

The basic assumption is that a service endpoint handles both the business messages and protocol messages (i.e. keys, evidences etc). As already mentioned, our prime objective is to free the service endpoints from performing security related tasks. Here, we will apply the same principle to design and implement a fair non-repudiation protocol. Although in [Agreiter et al. 2008] a first step has been achieved by integrating non-repudiation communication into Web service communication, there exists only a logical separation between non-repudiation messages and business messages. In this section we explain how these messages can be separated from business messages, by executing an out-of-band non-repudiation protocol. The proposed protocol does not only separate the business and security messages, but also maintains the desired properties of non-repudiation i.e. fairness and timeliness. In the next section, we will illustrate, how we design the non-repudiation protocol in the SeAAS architecture.

5.1 Designing Fair Non-repudiation protocol in SeAAS architecture

There are two different approaches to execute a NR protocol. The protocol is either enforced by the service endpoint [Zimmermann 2005] – in which case the service endpoint has to handle the security protocol messages in addition to business messages – or the service endpoint delegates the responsibility of executing the protocol to a dedicated non-repudiation service. The SeAAS architecture leverages the second approach.

In this approach, the protocol executes out-of-band between two dedicated non-repudiation services. The result is then communicated to the service endpoints, as shown in Figure 3. The NR services in domains 1 and 2 execute the protocol on behalf of the GP's *Client Application* (used by GP) and the ELGA service endpoint. Both delegate the security task to NR services through their respective SeAAS Engines. The detailed message communication to achieve fair non-repudiation in SeAAS architecture is shown as a UML Sequence Diagram in Figure 3. It shows the inter-domain communication between two non-repudiation services of the domains 1 and 2 and intra-domain communication among various components of each domain. The protocol is based on the ZG's protocol, which ensures *Fairness* and *Timeliness*.

The request to access the medical service is sent by a GP, through a *Client Application*. The ESB in domain1 intercepts the request and forwards it to the service endpoint of domain 2. The ESB in domain2 receives the request and routes it to the SeAAS Engine for security evaluation. The SeAAS Engine retrieves the policy that applies to the request from the Policy Repository and assigns the task to NR Service. Further security communication will take place among the non-repudiation services of two domains based on the detailed non-repudiation policy (an example is given in Section 6).

Non-repudiation is achieved by exchanging the evidences of messages sent and received: NRService@Domain2 requests NRService@Domain1 for the evidence of the service request sent by GP (Mess. 5). NRService@Domain1 retrieves the request details from ESB through SeAAS Engine (Mess. 6-9), signs the message and sends the signed message NRO1 to NRService@Domain2 as evidence. This message (Mess. 10) consists of a URI's of NRServices@Domain1 & 2 (represented by letters A & B respectively), timestamp (i.e. T), Label for any other information (i.e. L) and NRO1(i.e. evidence). Note, unlike ZG's fair Non-repudiation Protocol, we don't send the service request in

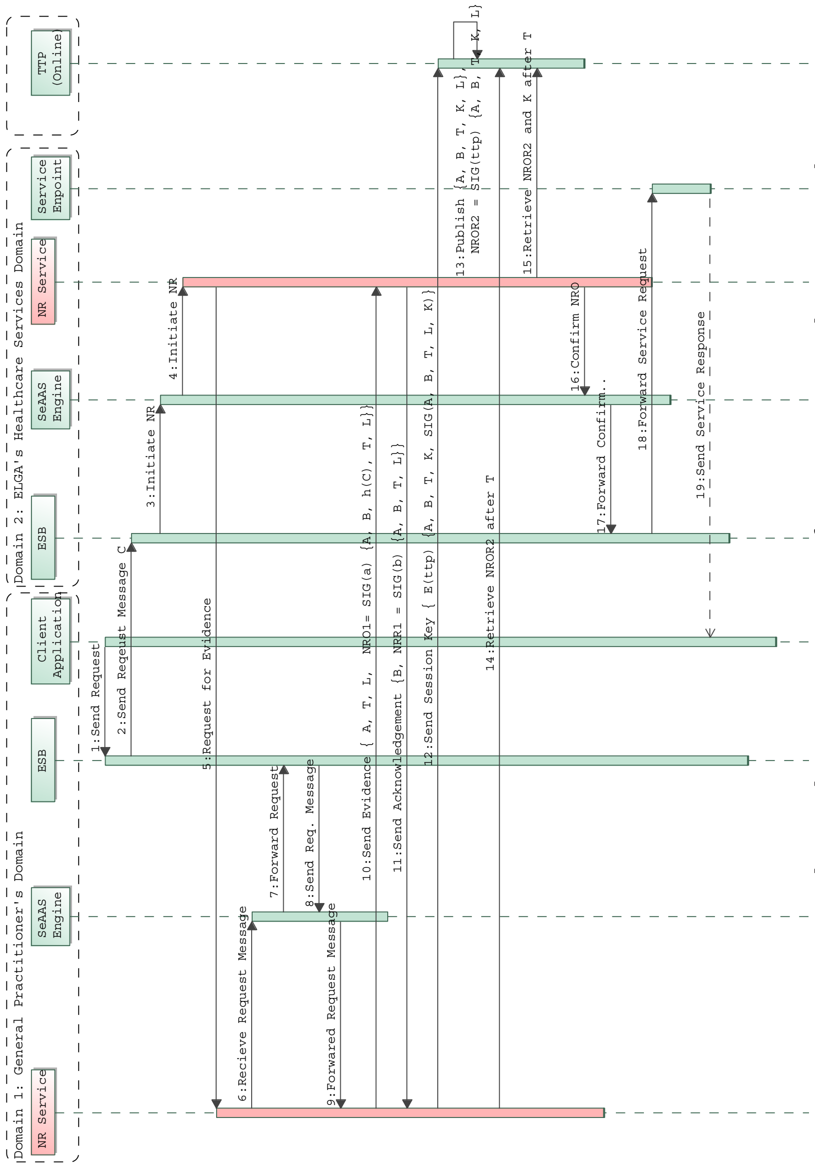
Domain 1: General Practitioner's Domain
NR Service
SeAAS Engine
ESB
Client Application
Domain 2: ELGA's Healthcare Services Domain
ESB
SeAAS Engine
NR Service
Service Enpoint
TTP (Online)
1:Send Request
2:Send Reqeust Message C
3:Initiate NR
4:Initiate NR
5:Request for Evidence
6:Recieve Request Message
7:Forward Request
8:Send Req. Message
9:Forwared Request Message
10:Send Evidence { A, T, L, NRO1= SIG(a) {A, B, h(C), T, L}}
11:Send Acknowledgement {B, NRR1 = SIG(b) {A, B, T, L}}
12:Send Session Key { E(ttp) {A, B, T, K, SIG(A, B, T, L, K)}
13:Publish {A, B, T, K, L}, NROR2 = SIG(ttp) {A, B, T, K, L}
14:Retrieve NROR2 after T
15:Retrieve NROR2 and K after T
16:Confirm NRO
17:Forward Confirm..
18:Forward Service Request
19:Send Service Response

Figure 3: Inter- and Intra-domain communication for Fair NR Protocol in SeAAS Architecture.

this message. Because, a service request is a business message, which has already been sent to service endpoint before beginning the NR protocol (Mess. 2). However, the encrypted message C requires a key i.e. K, and so far, the NRService@Domain1 has not sent that key to NRService@Domain2 for decrypting the request message.

To continue the protocol, NRService@Domain2 stores the evidence and sends a signed acknowledgment to NRService@Domain1. This is shown in Message 11 as Non- repudiation of Receipt (NRR1). At this moment, both NR services have the first part of evidence. The second part of the evidence will be signed by the TTP. Therefore, in Message 12, NRService@Domain1 sends the key K to TTP. TTP publishes the key and the second piece of evidence i.e. NROR2. Both the NR services retrieve this piece of evidence after time T. This completes the fair non-repudiation protocol between the NR services. After successful completion, the NRService@Domain2 sends the decrypted message and protocol completion notification to the service endpoint through SeAAS Engine (Messages 16-18). The service endpoint then sends the response to the client application. Thus, the protocol has executed out-of-band and the service endpoints were never involved. With this, we have not only separated the security from the business components in the architecture, but also the security communication from business communication.

6 Reference Architecture

The *Reference Architecture* (RA) is shown in Figure 4 as an UML deployment diagram. It shows components deployed, their relationships and Web services standards used. A service requester uses a *Client Application* to access the healthcare services offered by ELGA. The healthcare services are deployed at *Healthcare Systems Application Server*. The *ESB Server* provides a message-oriented middleware, which uses a *Dispatcher* component for inter- and intra-domain communication. The *SeAAS Server* hosts a *SeAAS Engine* component, which evaluates security based on the policy retrieved from the *Policy Repository*. The SeAAS Engine delegates the security task to security components[4], which are deployed at the *Security Server*. The Security Server deploys a number of security components i.e. Authentication, Non-repudiation, Authorization etc. The security components are configured at deployment time based on Service Component Architecture (discussed below) and the configurations are stored in a *SCA deployment Configurations* file. The security components use different Web services security standards for performing security tasks.

Policy assertions for functional and non-functional requirements are defined with **WS-Policy** [WS-Policy 2006]. For example, the Non-repudiation component specifies the specific policy describes supported protocols, types and contents of the evidence and cryptographic methods required for message protection. Figure 5 shows an example non-repudiation policy. It describes policy requirements as WS-Policy Assertions.

[4] We use the terms *Services* in the Conceptual Architecture (Fig. 2) and *Components* in the Reference Architecture (Fig. 4): a service is an abstract concept implemented by a component.

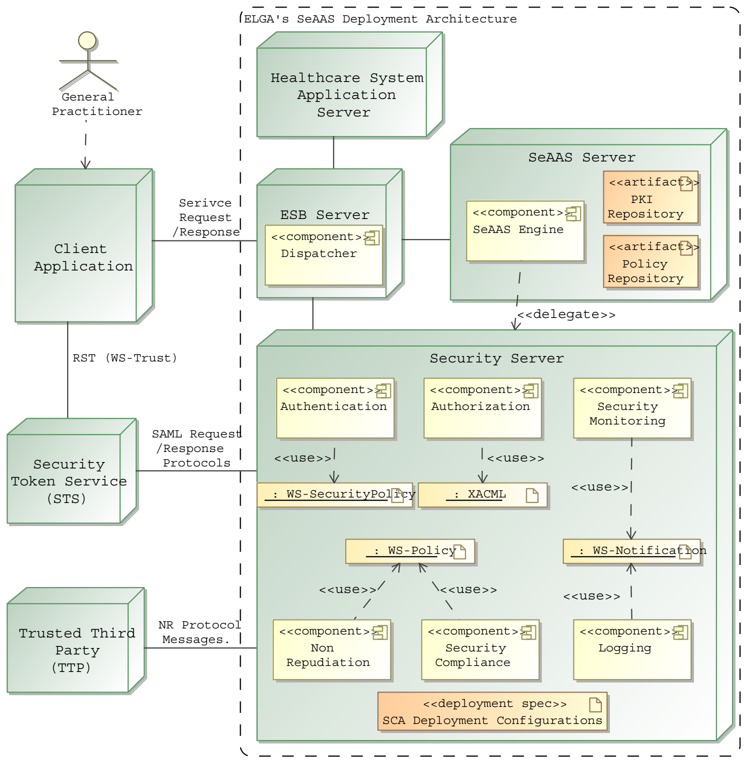
ELGA's SeAAS Deployment Architecture
General Practitioner
Healthcare System Application Server
Client Application
Serivce Request /Response
ESB Server
<<component>> Dispatcher
SeAAS Server
<<component>> SeAAS Engine
<<artifact>> PKI Repository
<<artifact>> Policy Repository
<<delegate>>
RST (WS-Trust)
Security Server
<<component>> Authentication
<<component>> Authorization
<<component>> Security Monitoring
Security Token Service (STS)
SAML Request /Response Protocols
<<use>>
: WS-SecurityPolicy
: XACML
: WS-Policy
: WS-Notification
Trusted Third Party (TTP)
NR Protocol Messages.
<<component>> Non Repudiation
<<component>> Security Compliance
<<component>> Logging
<<deployment spec>> SCA Deployment Configurations

Figure 4: The Reference Architecture.

The *Evidence Type* assertion defines that *DigitalSignature* on the message is required as an evidence. *Elements of Evidence* assertion defines that what should be the contents of a non-repudiation evidence. This implies that an evidence should contain the *MessageWithToken*, *MessageTimeStamp*, *URI's of EvidenceOriginator/ EvidenceRecipient* and *EvidenceExpiry*. The *ProtocolType* assertion defines that a *Fair Non-repudiation Protocol* is required which involves an *Online TTP*, defined in the *TTPRole* assertion.

Security requirements of a service endpoint are defined as security assertions embedded into WS-Policy assertions **WS-SecurityPolicy**[WS-SecurityPolicy 2007]. We use this standard to write the security policy of an endpoint, which defines supported type of bindings, tokens, encryption/signature algorithms. Two of the security components (security compliance and authentication) deployed at the Security Server use the WS-SecurityPolicy standard. The security compliance service checks the service request's compliance with the security policy. After the check, the authentication service proceeds for token validation for which it sends the requester's credentials to the Security Token Service (STS).

The **Security Assertion Markup Language** (SAML) is used to exchange security

```
<wsp:Policy wsu:id="NR_policy">
  <wsp:ExactlyOne>
    <wsp:All>

<!-- Evidence Type Assertion: -->
 <nrp:EvidenceType>
   <wsp:Policy>
     <nrp:DigitalSignature/>
    </wsp:Policy>
</nrp:EvidenceType>

<!---Elements of Evidence Assertion-->
<nrp:EvidenceElements>
  <wsp:Policy>
     <nrp:MessageWithToken/>
     <nrp:MessageTimeStamp/>
    <nrp:EvidenceOrginator/>
    <nrp:EvidenceRecipient/>
    </nrp:EvidenceExpiry>
  </wsp:Policy>
</nrp:EvidenceElements>
. . . .
```

```
 . . . .
<!-- NR Protocol Type Assertion-->
<nrp:NRProtocolType>
  <wsp:Policy>
   <nrp:FairNRProtocol/>
  </wsp:Policy>
</nrp:NRProtocolType>

<!-- TTP Role for NR Assertion-->
<nrp:TTPRole>
  <wsp:Policy>
      <nrp:onlineTTP>
      <nrp:TTP_uri uri="ttp01.org"/>
    </wsp:Policy>
  </nrp:TTPRole>

    </wsp:All>
  </wsp:ExactlyOne>
</wsp:Policy>
```

Figure 5: Example Non-Repudiation policy written in WS-Policy standard.

information between security domains [SAML 2005]. In the RA, two components of Security Server i.e. Authentication and Authorization components, use SAML standard. The authentication component creates authentication request/response based on SAML protocols. Using these protocols, the *Security Token Service* (STS) validates the tokens and sends a signed authentication SAML assertion to the authentication service, which forwards them to the service endpoint through SeAAS Engine. The Authorization component uses SAML in a similar manner. It makes authorization decision according to service requester's authorization policy and sends the decision as SAML authorization assertions to the service endpoint for enforcement.

The **Extensible Access Control Markup Language** (XACML) is a standard for authorization policies [OASIS 2006]. We use it to define permissions of a service requester. The *Policy Decision Point* (PDP) of the authorization service, makes decisions based on the permissions assigned to the roles (e.g. practitioner), defined as XACML rules.

WS-Trust provides interfaces for token issuance and validation [WS-Trust 2005]. A service consumer can get security tokens from an STS. The authentication component uses the WS-trust interface to get token validation decision by STS.

We use **WS-Notification** to send event notifications to Logging and Security Monitoring components [WS-Notification 2006]. Logging notifications carry information pertaining to the service requests and responses, whereas security alerts are notifications for security monitoring.

The **Service Component Architecture (SCA)** model is used for composition of security services performed by SeAAS Engine [SCA 2007]. Security components are integrated to a *Security Composite*, which realizes a set of security requirements. SCA composite is written in the XML-based *Service Composition Definition Language*. The component-based architecture facilitates independence from language and technology, reusability, and improves extensibility and maintainability. We use SCA properties for configuration during deployment of security components based on security policy of a domain. These configurations are stored in *Deployment Specifications* as an *SCA Deployment Configurations* file, as shown in Figure 4.

The **Lightweight Directory Access Protocol** (LDAP) is used for directory access to retrieve certificates, policies and related information [IETF 2006]. The policy repository is used for storage and retrieval of security policies, Authorization policies, and component policies like e.g. Non-repudiation used by SeAAS Engine and the Security services. The PKI Repository holds certificates and keys of service endpoints.

7 Conclusion and Future Work

In this paper, we presented an architecture that realizes Security as a Service. We motivated our approach with a discussion of the many limitations of endpoint security, the current practice in SOA security to enforce security with the endpoints. By default, the engineering intuition seems to impose a turning away from the concept of centralization entailing the threat of a single-point-of-failure or unbearable communication overhead. But we showed how concepts of SOA, like declarative security, the Enterprise Service Bus and Model Driven Security, an advanced method of software engineering can open a new venue to the efficient realization of security critical, inter-organizational processes. The reference architecture is able to cope with the complex security requirements imposed by use cases from industries that have to deal with security-critical processes spanning multiple security domains.

The objection that SeAAS creates a single point of failure or a bottleneck can be countered by balancing the workload over replicated service components – an inherent advantage of SOA-style architectures. Taking communication overhead into account it does not always make sense to outsource all security functionality to a SeAAS provider. Tasks like basic XML processing can be left with the endpoint. It is up to the architect to decide upon the degree of service centralization. A hybrid approach distributes the tasks between endpoints and the SeAAS according to specific need. We currently envisage the extension of our architecture to support the flexible, run-time adjustment of the degree of centralization. Security services are registered with the SeAAS engine and advertised to potential consumers. Authorized requesters would access them as needed.

In this paper, we used Non-repudiation as an example for Protocol Based Security. We are currently adding service components to cover all requirements as introduced in Section 2.2. The integration of functionality enforcing Auditing Policies for compliance to regulatory and corporate requirements necessitates more conceptual work as deviant behavior and harmful incidents (e.g., fraud) can only be detected in context of application and process level semantics. We are also actively working on the systematic and comprehensive definition of extensions to WS-Policy and WS-SecurityPolicy in order to support the declarative specification of complex security requirements configuring the SeAAS engines distributed over the various security domains.

References

[Agreiter et al. 2008] Agreiter, B. Hafner, M. Breu, R.: “A Fair Non-repudiation Service in a Web services Peer-to-peer Environment”; Computer Standards and Interfaces, 30(6), 372–378, 2008.

[Alam et al. 2007] Alam, M. Hafner, M. Memon, M. Hung, P.: “Modeling and Enforcing Advanced Access Control Policies in Healthcare Systems with SECTET”; In MOTHIS ’07: MODELS 2007, Nashville, USA, 2007.

[Arge ELGA] Arge ELGA: Arbeitsgemeinschaft Elektronische Gesundheitsakte; `http://www.arge-elga.at/`

[Atkinson et al. 2002] Atkinson, B. et al.: “Web Services Security (WS-security) - version 1.0. Specification; IBM Corp., Mircosoft Corp., VeriSign, Inc., 2002.

[Bajaj 2006] Bajaj, S.: “Web Services Policy 1.2 - Framework (WS-Policy)”, W3C Member Submission; 25 April 2006, Technical Report, W3C, 2006.

[Bartel et al. 2002] Bartel, M. Boyer, J. Fox, B.: “XML-Signature Syntax and Processing”; W3C Recommendation, 12 February 2002, Technical Report, W3C, 2002.

[Basin et al. 2006] Basin, D. Doser, J. Torsten, L.: “Model Driven Security: From UML Models to Access Control Infrastructures”; ACM Trans. Softw. Eng. Methodol., 15(1), 39–91, 2006.

[Becker and Sewell 2004] Becker, M. Y. Sewell, P.: “Cassandra: Flexible Trust Management, Applied to Electronic Health Records”; In Comp. Sec. Foundations Workshop, 2004. Proc. 17th IEEE, 2004.

[Donner 2004] Donner, M.: “From the Editors: Whose Data are These, Anyway?”; IEEE Security and Privacy, 2(3), 5–6, 2004.

[Hafner and Breu 2008] Hafner, M. and Breu, R.: “Security Engineering for Service-oriented Architectures”; Springer, October 2008.

[Hafner et al. 2006] Hafner, M. Breu, R. Agreiter, B. Nowak, A.: “SECTET: An Extensible Framework for the Realization of Secure Inter-organizational Workflows”; Journal of Internet Research, 16(5), 491-506, Emerald, 2006

[Hinton et al. 2005] Hinton, H. Hondo, M. Hutchison, B.: “Security Patterns within a Service-oriented Architecture”; Nov. 2005, `http://www.ibm.com/websphere/developer/services`.

[Hohpe and Woolf 2004] Hohpe, G. Woolf, B.: “Enterprise Integration Patterns : Designing, Building, and Deploying Messaging Solutions”; Addison-Wesley Professional, 2003.

[IETF 2006] IETF.: “The LDAP Technical Specification”; 2006, `http://tools.ietf.org/html/rfc4510`.

[Imamura 2002] Imamura, T.: “XML Encryption Syntax and Processing”; W3C Recommendation, 10 December 2002, Technical Report, W3C, 2002.

[InnoQ 2006] InnoQ: “Web services Standards Overview”; Nov. 2006, `http://www.innoq.com/soa/ws-standards`

[Josuttis 2007] Josuttis, N. M.: “SOA in Practice: The Art of Distributed System Design”; O’Reilly Media, Inc., August 2007.

[Juerjens 2004] Juerjens, J.: “Secure Systems Development with UML”; SpringerVerlag, 2004.

[Kanneganti and Chodavarapu 2007] Kanneganti, R. Chodavarapu, P.: “SOA Security in Action”; Manning Publications Co., Greenwich, CT, USA, 2007.

[Kremer et al. 2002] Kremer, S. Markowitch, O. Zhou, J.: “An Intensive Survey of Fair Non-Repudiation Protocols”; Computer Communications, 25, 1606–1621, 2002.

[Lopez et al. 2005] Lopez, J. Montenegroa, J. Vivasa, J. et. al.: “Specification and Design of Advanced Authentication and Authorization Services”; Computer Standards and Interfaces, 27(5), 467–478, 2005.

[Markowitch et al. 2001] Markowitch, O Kremer, S.: “An Optimistic Non-repudiation Protocol with Transparent Trusted Third Party”; In ISC ’01: Proc. of the 4th Int. Conf. on Information Security, London, UK, Springer, 2001.

[Markowitch et al. 1999] Markowitch, O. Roggeman, Y.: “Probabilistic Non-repudiation Without Trusted Third Party”; In 2nd Conf. on Security in Communication Network, 1999.

[Memon et al. 2008] Memon, M. Hafner, M. Breu, R.: "SECTISSIMO: A Platform-Independent Framework for Security Services"; In ModSec '08: MODELS 2008, Toulouse, France, 2008.
[McAfee 2008] McAfee.: "Security as a Service"; 2008, `http://www.mcafee.com/us/local_content/solution_briefs/sb_saas_0709.pdf`
[Microsoft 2006] "Microsoft Windows live onecare"; 2006, `http://onecare.live.com/standard/enus/3/default.htm`.
[OASIS 2006] OASIS TC.: "Extensible Access Control Markup Language (XACML)"; 2006. `http://www.oasis-open.org`
[ORACLE 2008] Oracle.: "Service-Oriented Security: An Application-Centric Look at Identity Management"; 2008, `http://www.oracle.com`.
[Peterson 2009] Peterson, G.: "Service-oriented Security Indications for Use"; IEEE Security and Privacy, 7(2), 91–93, 2009.
[Peterson 2005] Peterson, G.: "Service-oriented Security Architecture"; 2005, `http://www.arctecgroup.net/ISB1009GP.pdf`
[Rademakers and Dirksen 2008] Rademakers, T. Dirksen, J.: "Open-Source ESBs in Action"; Manning Publications Co., Greenwich, CT, USA, 2008.
[SAML 2005] OASIS TC.: "Security Assertion Markup Language (SAML)"; 2005, `http://www.oasisopen.org`.
[Satoh et al. 2006] Satoh, F. Nakamura, Y. Ono, K.: "Adding Authentication to Model Driven Security"; In ICWS '06: Proc. of the IEEE Intern. Conf. on Web Services, Washington, DC, USA, 2006.
[SCA 2007] Chappell, D.: "Introducing SCA"; 2007, `http://www.davidchappell.com`.
[ServiceMix 2008] "Apache-ServiceMix: An Open Source ESB"; `http://servicemix.apache.org`
[Symantec 2006] Symantec. Symantec names Genesis Norton 360; 2006, `http://www.symantec.com/about/news/release/article.jsp?prid=20060531_01`.
[WS-Notification 2006] OASIS.: "Web Services Notification (WSN) Specifications"; 2006, `http://docs.oasisopen.org/wsn`.
[WS-Policy 2006] W3C.: "Web Services Policy 1.2 - Framework"; 2006, `http://www.w3.org/Submission/WSPolicy`.
[WS-SecurityPolicy 2007] OASIS TC.: "WS-SecurityPolicy"; 2007, `http://docs.oasis-open.org`.
[WS-Trust 2005] OASIS TC.: "WS-Trust Sepcifications"; 2005, `http://docs.oasis-open.org`.
[Zhou et al. 1999] Zhou, J. Deng, R. H. Bao, F.: "Evolution of Fair Non-repudiation with TTP"; In ACISP '99: Proc. of the 4th Australasian Conf. on Information Security and Privacy, London, UK, 1999. Springer.
[Zimmermann 2005] Zimmermann, R.: "Design and Prototypical Implementation of a Non-Repudiation System for Mobile Grid Services"; `http://www.ifi.uzh.ch/archive/mastertheses`.