FAKULTÄT
FÜR !NFORMATIK
Faculty of Informatics

Security for the Integration of Software Tools in Multidisciplinary Engineering Processes

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieur

im Rahmen des Studiums

Software Engineering & Internet Computing

eingereicht von

Christoph Gritschenberger
Matrikelnummer 0525747

an der
Fakultät für Informatik der Technischen Universität Wien

Betreuung: Ao. Univ.-Prof. Dr. Stefan Biffl
Mitwirkung: Dipl.-Ing. Dietmar Winkler

Wien, 21.11.2011

(Unterschrift Verfasser/in)

(Unterschrift Betreuung)

Erklärung zur Verfassung der Arbeit

Christoph Gritschenberger
Aspersdorferstraße 8, 2020 Hollabrunn

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit - einschließlich Tabellen, Karten und Abbildungen -, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

(Ort, Datum) (Unterschrift Verfasser/in)

Abstract

Software Engineering has become an integral part of many engineering processes in various domains. Software is used to control the behavior of complex systems in production, but is also used for support or control during development processes. The term (Software+) Engineering is used to describe such a software-intensive heterogeneous engineering environment. To address the challenge of integrating the variety of tools and domains involved the "Open Engineering Service Bus" was constructed to specifically suite the needs for (Software+) engineering with the goal to improve collaboration across engineering domains.

With increasing complexity of projects and the growing knowledge in many domains, projects are separated to smaller components which are then taken over by another party. With companies specializing on specific domains and the increased need to outsource tasks, security concerns when using software for collaboration become inherent. Data and infrastructure is shared on the Internet exposing it to a wide range of threats.

In a project every party has assets that need to be protected from access or modification by another party. For this to work a common model for exchanging data and other queries is established. The need for all parties to authenticate is considered in this model. This enables all parties to implement access control into their infrastructure. However relying on tool specific security implementations for access control makes it hard to exchange tools in a process or reuse processes in other projects. So an additional abstraction layer for access control is required to decouple it from specific tool implementations.

Based on a real world scenario where electrical planning tools from different vendors are integrated across two companies, usecases are specified with security in mind. Possible threats are identified and components for mitigating the threats are designed. They are integrated with the tool connector infrastructure and made transparent to other components. A prototype is implemented based on the OpenEngSB research prototype. The solution is validated by designing integration and system tests that are based on specified usecases.

The main components are an Authentication and Authorization component and integration of these security concepts in distributed environments. The solution is suitable for implementing complex access control structures. The constructed policies can be designed in a central place and are independent of the specific tools used. The integration is mostly seamless and hardly affects the usability for developers and management, but has a noticeable impact on performance. With the additional access control layer the tools remain completely exchangeable while all interactions across networks are secured against the most common threats.

Kurzfassung

Software Engineering ist zu einem integralen Bestandteil vieler Engineering-Prozesse in verschiedenen Bereichen geworden. Software wird verwendet, um das Verhalten komplexer Systeme sowohl im Produktionsbetrieb als auch im Entwicklungsprozess zu steuern. Der Begriff (Software+) Engineering wird verwendet, um eine solche software-intensive heterogene Engineering-Umgebung zu beschreiben. Um sich der Herausforderung der Integration der Vielzahl von Werkzeugen und Domänen anzunehmen, wurde der "Open Engineering Service Bus" speziell auf die Bedürfnisse von (Software +) Engineering konstruiert, um die Zusammenarbeit zwischen Engineering-Domains zu verbessern.

Mit zunehmender Komplexität der Projekte und dem wachsenden Wissen in vielen Bereichen, werden Projekte in kleinere Komponenten getrennt, die dann von verschiedenen beteiligten Parteien abgewickelt werden. Aus diesem Grund wird die Sicherheit der Software bei der Zusammenarbeit zuhehmend wichtiger. Daten und Infrastrukturen die über das Internet verteilt werden, sind einem breites Spektrum von Bedrohungen ausgesetzt.

Jede Partei in einem Projekt verfügt über Resourcen, die vor Zugriff von einer anderen Partei geschützt werden müssen. Ein gemeinsames Datenmodell, das es alle Beteiligten erlaubt einander zu authentifizieren wird entwickelt, um jeder Partei die Entwicklung eigener Zugriffskontrollen zu ermöglichen. Die Auslagerung dieser an werkzeugspezifische Implementierungen macht es schwierig, Prozesse in anderen Projekten mit unterschiedlichen Werkzeugen wiederzuverwenden. Eine zusätzliche Kontrollschicht ist erforderlich, um diese von speziellen Implementierungen zu entkoppeln.

Basierend auf einem realen Szenario, in dem elektrische Planungstools von verschiedenen Anbietern über zwei Unternehmen integriert sind, werden Anwendungsfälle unter Berücksichtigung von Sicherheitsaspekten festgelegt. Mögliche Bedrohungen werden identifiziert und Komponenten zur Beseitigung dieser werden erstellt. Sie sind mit der Infrastruktur integriert und für andere Komponenten transparent. Ein Prototyp wird auf der Grundlage der OpenEngSB Forschungsprototyps implementiert. Die Lösung wird durch Integrations- und Systemtests, die auf den Anwendungsfällen basieren, validiert.

Die Hauptkomponenten sind eine Authentifizierungs- und eine Autorisierungskomponente sowie deren Integration in verteilten Umgebungen. Die Lösung eignet sich für die Umsetzung komplexer Zutrittskontrollstrukturen. Diese können an einem zentralen Ort und unabhängig von spezifischen Werkzeugen verwaltet werden. Die Integration ist meist nahtlos und hat kaum Auswirkungen auf die Nutzbarkeit für Entwickler und das Management, jedoch aber einen Einfluss auf die Leistung. Mit der zusätzlichen Zugriffskontrollschicht sind Werkzeuge austauschbar, während alle Interaktionen in Netzwerken gegen die häufigsten Bedrohungen gesichert sind.

Contents

CHAPTER 1

Introduction

Software tools are used to improve process efficiency in many domains. Not only in software engineering, but also automation engineering and electrical engineering software is used to ease development and maintain project related artifacts. Such software intensive engineering environments are summed up under the term (Software+) Engineering.

Integrating heterogenous software systems in enterprise context is commonly refered to as Enterprise Application Integration (EAI). So first EAI and the concept of the Enterprise Service Bus (ESB) are introduced in the next chapter. The OpenEngSB software platform is built upon the concept of an ESB and aims to satisfy requirements in a (Software+) Engineering context. So after that the origin and purpose of the OpenEngSB platform are presented.

1.1 The Engineering Service Bus Concept

During development of software-intensive systems, expertise from several engineering disciplines may be required. Each expert prefers a specific set of tools suitable for the respected domain. Also they will prefer using tools they are familiar with. As a result, many different software tools are used throughout a project. In general most of these tools may only be compatible to specific other tools or not compatible at all. There are several approaches to integrate tools in enterprise environments. These approaches are developed under the term Enterprise Applicaction Integration (EAI).

Of course in (Software+) engineering processes artifacts of different domains are likely to have some intersection point. Some artifacts depend on other artifacts. These dependencies sometimes also cross domain boundaries. For example when designing a controller curcuit for a machine, there is often some software involved interfacing with the controller. This means that parts of the curcuit have some connection to elements in the software. Though maintaining consistency is often critical, software tools often do not support interaction with tools from other domains (e.g. link an UML-diagram to a circuit diagram). So these dependencies must be maintained in a way, that developers can easily spot them and adapt the design accordingly. Also

developers must be notified when a connected artifact is changed. Moreover the dependencies should be made aware to the project management to track the progress.

Like an Enterprise Service Bus (ESB) [15] the engineering service bus is intended to serve as a bridge crossing organizational and geographical boundaries. In addition the integration of data of engineering artifacts is considered. While in software engineering there are standardized formats for design documents like e.g. the Unified Modeling Language (UML)[1], such standards are quite rare in other engineering domains (automation engineering, electrical engineering). But even in software engineering with help of these standards, data integration issues remain.

Biffl et al [10] extend the ESB-concept by additional approaches for integrating development tools and their data. As opposed to integration in Business IT the software tools used in engineering environments often focus on specific tasks for a single user while not providing APIs to integrate into an enterprise environment. So it may be difficult for engineering tools to be integrated in heavy-weight middleware because they cannot support the finegrained interfaces used. Also systems are always assumed online, while engineers may go offline at any point in time [10].

1.2 Security Motivation

While in small groups universal access to all resources may be sufficient, with growing size of projects and increasing number of people working together the need for security is imminent. Access to resources should be limited to project and organization members that are authorized. So when securing this environment, policies must be designed and enforced. These policies are designed to protect organization assets from unauthorized employees. Moreover when experts of multiple domains are involved, the data integration must be extended to external companies. Exposing services and data to external entities must be done with security in mind. Services must be prevented from abuse, and data must be protected not only from public access but also from unauthorized personel. This creates the need for quite complex access control structures.

Companies may want to employ security policies that are not bound to a specific software programs in their development process because it provides the possibility to exchange software used in their processes. This concept on the one hand needs to be flexible to suite requirements by enterprises. On the other hand it must be simple for companies to integrate it in their workflow and add support to their tools. Security policies need to be enforced across company and domain boundaries.

Security considerations are triggered and influenced and by a variety of steakholders and experts. The following three sources are considered the most important sources of security considerations.

Management

(Software+) engineering workflows require many interactions among developers and managers. This includes interactions on different management levels as well as across engineering discipline boundaries. Every project member is responsible for specific aspects of the project.

[1] http://www.omg.org/spec/UML/

In small projects it is often sufficient to provide a shared directory on an company internal network with public access. Processes are documented on paper and workflows are managed by simply writing emails and hold project meetings. However, with growing project size, these solutions need to be refined or replaced. The network directory may be replaced by version control system. Processes are defined using specific modeling tools. Holding project meetings with all members present becomes unfeasible.

So project management tools are needed to distribute instructions and track the progress. Integrated management solutions require interactions from every project member. Obviously it is not appropriate to allow any action to every employee. Allowed interactions depend on the member's role in the project. For example a developer should not be allowed to announce meetings with the customer.

Suppose an employee is not supposed to perform certain actions in the project. Without automatic security barriers there is always the risk that he performs it anyway. This could even happen by accident, i.e. without bad intentions on the side of the employee.

But maintaining workflows is not the only security aspect to be concerned about. It is critical to control the access to certain project artifacts and documents. A developer should not be able to modify an artifact that he is not supposed to.

Threats

In general every person with access to the network could be the origin of an attack. When exchanging data over the Internet it must be assumed that any message can be viewed and altered by attackers. However attacks often originate from employees or former employees. They might possess insider knowledge making attacks easier. So employees should only be entrusted with access to the systems they are are supposed to. There are several ways in which an employee can be a threat. If an employee performs some unauthorized action this can be intentional or unintentional. Unintentional tempering is common in environments that do not enforce security policies. Protection from these unintentional breaches can be done with comparatively little effort. Sometimes these protections do not even involve real security, but just some additional barriers in the user interface. For example additional checkboxes or dialogs may distract the user enough to not perform the unauthorized action.

There may be a wide range of reasons for an employee to intentionally access some system or data he is not authorized to. The objectives of these attacks from within may include but are not limited to stealing or manipulating sensitive data and damaging service infrastructure. Of course employment also entails a certain degree of trust. It is important that employees are always able to do their job. Security systems should not restrict them from doing it.

But there is a variety of additional threats that must be considered. More threats can be identified using the STRIDE approach [36].

1.3 OpenEngSB architecture

The OpenEngSB is an implementation of the Engineering Service Bus concept described in 1.1. It serves as a framework for integrating tools in enterprise environments. The core of the

system is implemented in Java and uses an implementation of the OSGi-framework [58]. The OSGi specification provides a module system for managing software components as well as an infrastructure for registering and querying services. The initial architecture of the OpenEngSB is described by Pieber [49]. While the basic ideas are still valid many specific aspects of the described architecture have changed because many aspects related to the JBI specification [59] have been either removed or replaced with corresponding counterparts from the OSGi specification [58, 57]. Domains and connectors are no longer represented as JBI "service engines" but as OSGi "bundles". A more recent verion of the archtecture description is available on the project's homepage [2]

The aspects of the architecture relevant for security are described here.

Components and Services

The OSGi specification [58] states that "It provides a general-purpose, secure, and managed Java framework that supports the deployment of extensible and downloadable applications known as *bundles*." The OpenEngSB consists of several components. These components are delivered as separated OSGi-bundles. Bundles may be modified, updated at runtime without requiring a restart of the system [58]. Among many things bundles provide services. The bundle is responsible for registering and managing services. Services are exported exposing only their Java interfaces. This allows decoupling of interfaces and their implementations.

The process of registering a services involves the following parameters [58]:

- A Java Object representing an instance of the service's implementation
- One or more interfaces the service provides
- Properties in form of a Dictionary

The properties are a set of key/value pairs that represent information about the service. It is suggested to only used primitive or standard java types. There is also support for collections and arrays as values [58]. These properties can be used in service-filters to create more finegrained queries.

The OpenEngSB provides most of its core functionality as OSGi-services. This means that they are bound to the same properties as any OSGi-service.

- They are bound to the lifecycle of the bundle that registered it
- They are queried and accessed as any other service

Generally the core services are created and registered when their corresponding bundle is started.

Domains and Connectors

Pieber [49] introduces the concepts of "Tool Connectors" and "Tool Domains". Tool connectors are defined to "connect external tools in a protocol and platform independent approach to the

[2] http://openengsb.org

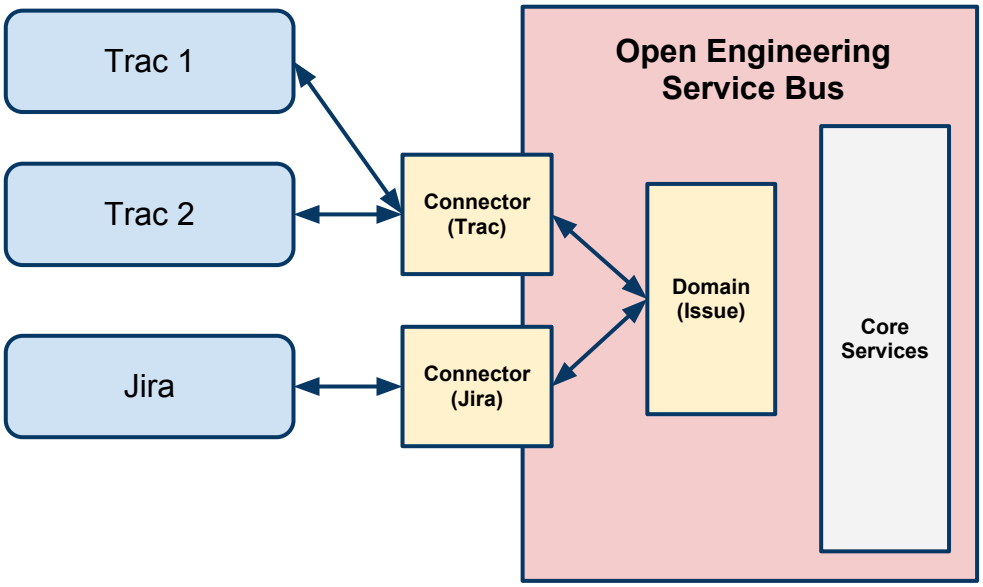

Figure 1.1: Architecture Overview of OpenEngSB (Examples are in parenthesis).

OpenEngSB". In this work, they are refered to as only "Connectors" because they are used to connect not only external tools, but also other external systems like different storage backends.

Tool Domains serve as an abstraction for a group of tools. They define common functionality, so that any of the Tool connectors in this group can be used in place where the use of the domain is specified. "Tool domains could be compared best to the concept of abstract classes in in [sic!] object orientated programming languages" [49]. Similar to the Connectors, the concept of Tool Domains is also used in a more generic way, thus refered to as just "Domains". Figure 1.1 illustrates the relation of tools, connectors and domains in the OpenEngSB architecture.

Domains are developed as OSGi-bundles and usually provide:

- A domain *interface* that specifies supported operations.
- A *common data model* that is used to specify common aspects of data generated by the abstracted systems.
- Utility code in form of abstract classes or utility classes useful to most connectors.

Connectors are also developed as OSGi-bundles. They consist of

- Connector implementation: A class implementing one ore more domain interfaces. A "Connector instance" is usually an instance of this class registered as a service.
- Connector factory: A factory implementation that is responsible for creating new connectors and applying configurations to an instance.

Context

The concept of a "context" is considered and present throughout the architecture of the OpenEngSB. It is used to separate organizational units. Example for such organizational units can be separate projects. Most operations that are performed in an OpenEngSB environment are executed in a context. Workflows and connector implementations are intended to be reused. The context satisfies the need for project-specific configuration. Workflows are specified in an abstract manner, so they only reference external services as domains and do not require specific connector implementations. In order to create a service composition to be used in a workflow in a specific context, the services are wired using service properties.

Remote Access

An OSGi-container only provides a local service registry that are not accessible through the network. The OSGi-specification covers a specification for "Remote services", but this specification is designed for interoperability between multiple OSGi-containers. In general external tools and client connectors do not necessarily run in OSGi-containers. A generic infrastructure for remote access is used in the OpenEngSB in order to be able to interact with external systems that do not run on the same machine. It also allows interoperability with components that are not written in Java (e.g. remote clients implemented in C#).

The network communication is performed by exchanging selfcontained messages, thus following a stateless concept. Every sent message can optionally yield a response to handle results and errors. There is no specific format or protocol defined for exchanging these messages. The actual network communication is handled by components called "ports" which are provided as OSGi-services as well. The ports are responsible for transporting service requests. This means ports cover all aspects of the communication itself (e.g. establishing connections, handshakes, etc.). The processing of the requests is handled a core component called "Request Handler".

Workflow

"The workflow component is responsible for engineering rule, process and event management in the OpenEngSB" [49]. The workflow component of the OpenEngSB uses the "Drools - The Business Logic integration Platform" [3] to provide this functionality. It is responsible for managing rules and workflow definitions to handle events. An important part of the workflow integration is the wiring of workflow actions to services in the OpenEngSB. Following the domain architecture, workflows are intended to be designed using references to domains only. This way a workflow definition can be reused in multiple contexts that may not even use the same set of tools. The wiring takes into account that a service can be unregistered or updated at any time. So instead of binding the service once at startup it is resolved ad-hoc every time it is requested in a workflow.

Pieber [49] proposes to assign this task is assigned to a component called "registry". In the current version of the OpenEngSB OSGi itself is used as registry. To be able to bind connector-

[3] http://www.jboss.org/drools/

instances to workflows in a specific context, service properties are used. The registry provides the ability to manage properties of a service registration that can be queried.

Engineering Database (EDB)

The EDB component implements the concept proposed by Waltersdorfer et al [60]. It provides generic versioning storage for tool generated data. The base format is based on key-value-pairs. It serves as the core component for automatic data transformation.

1.4 Scope of the Thesis

Since providing a complete company security concept exceeds the scope of this thesis, some assumptions made. The focus is to securing the framework itself, so it is assumed that the machine(s) running the bus are secure. This means that there is no security issue like for example an outdated operating system. For reasons of simplification, it is also assumed that the machines running the bus are running no other software that can interfere with the functionality and security of the OpenEngSB (e.g. rootkits). Although in reality this is never the case, security issues that only occur in combination with other software will be omitted. Also external tools that operate out of the scope of the OpenEngSB are considered secure. The focus is on securing the interface between the external tool and the OpenEngSB.

Although the solutions sometimes employ constraints on encryption algorithms to symmetric or asymmetric, the solution is generally independent of any specific algorithm implementation. However specific algorithms are chosen to evaluate the solution. All algorithms used are assumed to be secure if not stated otherwise. Also cryptoanalysis using pattern matching or similar techniques is not a considered threat in this thesis.

All passwords and shared secrets are assumed to be sufficiently cryptographically strong. It should not be feasible to derive a password using brute force or guessing attacks.

So the focus is on securing the bus itself. The core assets that need protection are components provided as services.

Requests for services can originate from various sources:

- External messages: Requests that are forwarded to services may originate from some remote location. The request is transmitted via some arbitrary protocol in a protocol-specific message format.
- Internal invocations: During execution services may invoke other services in order to process a request.
- Web Interface: Although invocations are mostly treated like internal invocations, there are some special cases to consider in user interfaces (like hiding elements the user is not authorized to use).

Web applications highly depend on stateful sessions, and the only client software required is a browser. Most applications only require a single login, that is saved throughout a session. Further the login is often kept alive using cookies.

Message oriented Middleware is generally stateless and does not support sessions out of the box [42]. So a custom protocol for assign the messages to a context has to be established.

The goal is to reuse existing practices and standards where possible. There may be aspects in the system where applying standards is not feasible, which need to be identified.

CHAPTER 2

Related Work

In this section relevant related work is discussed. An overview of the OpenEngSB architecture and components was already presented in 1.3. In subsection 2.2 a more deep explanation is provided how the concepts may be used to provide tool integration solutions like required in the section before. Also the state with regard to security in that implementation is outlined.

2.1 Tool Integration in Automation Systems Engineering

In automation systems engineering (software+) engineering processes are used to improve productivity by early defect detection in software-intensive automation systems. Software engineers working in this context depend on inputs from other engineering disciplines. The EngSB concept is used to introduce improved tool integration in automation system environments. The integration is based on the "Automation Service Bus" concept [10].

Biffl et al [11] described a basic architecture for technical integration of security in distributed engineering processes (see Figure 2.1).

An example for a typical large-scale engineering project would be building a power plant. Expertise from multiple engineering disciplines is required to realize such a large-scale project.

Typically a broad range of engineering tools is involved in solving specific problems. Some vendors provide sets tools that are integrated with each other as suites. However these suites as well as other tools in most cases are not able to exchange data with tools from other vendors [10].

Figure 2.2 illustrates a typical layout for a project like a power plant. Engineers for the electrical, mechanical and software development domain are working together on a project. A common assumption is that each engineering team uses tools specifically suited for tasks in their domain. Typically each component must provide interfaces for other components to be connected. The development teams of each component need to collaborate on the desgin of these interfaces.

The common concept for all domains is the "signal". A signal is a part of the component that multiple domains need to interface with. For example, a valve (mechanical engineering)

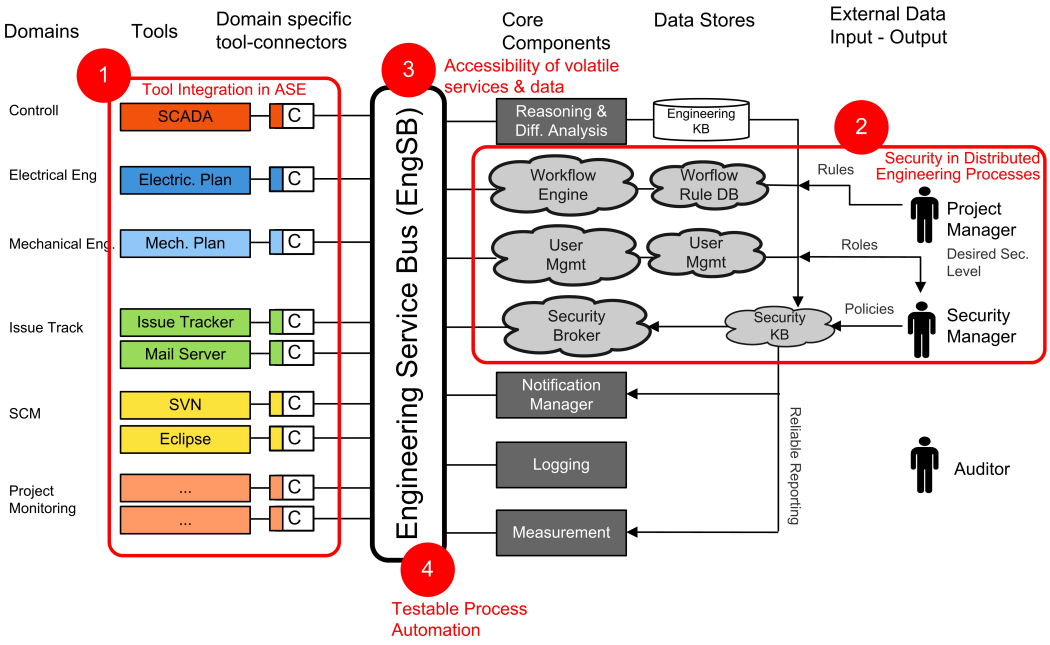

Figure 2.1: Current state and research challenges in Technical Integration and Security on the EngSB. [11]

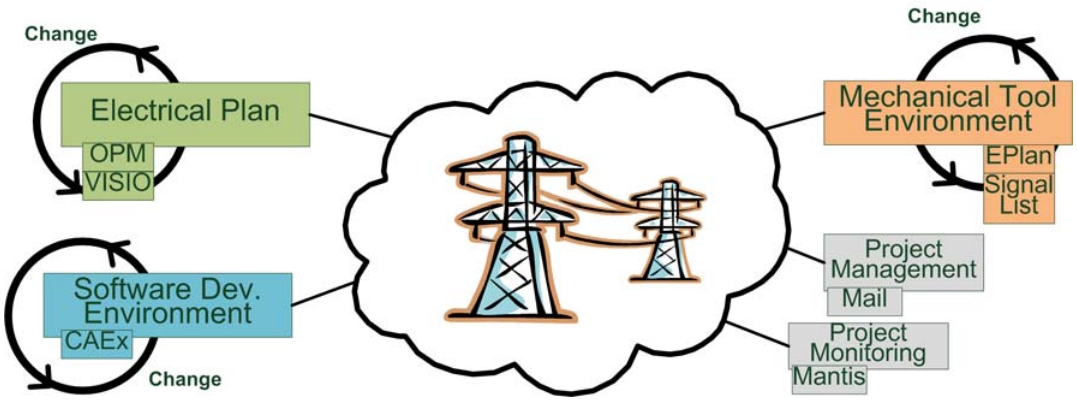

Figure 2.2: Tool setup in signal engineering [11]

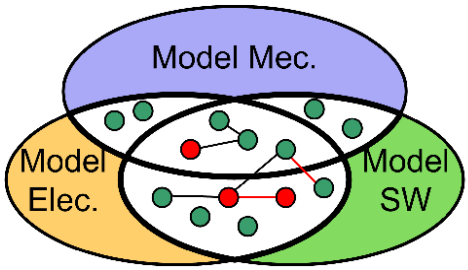

Figure 2.3: Domain tools with overlapping models [11]

is controlled by a PLC (electrical engineering) that as controlled by software. The state of the valve is relevant to all of the three domains. While electrical engineers like to think of it as input for the PLC, software developers may refer to the same signal as variable in the program.

The following tools are used in Figure 2.2:

EPlan EPlan is tool to create circuit diagrams, which are used for control cabinets manufacturing. It can generate a cable report that is used for wiring connections between control cabinets, sensors and consumers. The import and export format for Programmable Logic Controllers (PLC) is called ZULI [1]. The format is based on Comma Separated Values (CSV). Each signal description includes a technical address and a unique craft number. EPlan also provides an API to for external programs to interact with.

OPM OPM tool is used to map physical addresses of signals to virtual software addresses. It also may export data to a CSV based format including the technical address and the unique craft number.

CAEx CAEx is used for editing function block diagrams. It helps engineers to logics and control functions.

Mantis MantisBT web-based bugtracking system. It is used for tracking issues and changes to components.

The development process involves the following steps

1. requirements and specification
2. system typology (I/O cards and network typology)
3. curcuit diagram
4. hardware design and assembly
5. create PLC software

[1] German words ZUlieferungsLIste (ZULI) (supplier list)

6. rollout of the system

In general 25 % of all signals specified are changed in the life cycle of the development. Changing signals means that as a consequence other development of other domains working with the same signal maybe affected.

"Today's systems integration technologies in automation systems engineering are capable of integrating engineering tools from a range of engineering disciplines, but often rely on manual data processing and exchange."

Data is often processed manually and maintained in CSV formatted files. The fact that CSV is used as a common format has the advantage that no special tools are needed to view and edit the files. It is not uncommon that these files are maintained using spreadsheet applications like Microsoft Excel[2]. Sometimes the history of the files is preserved by using version control systems like Subversion[3].

Data is exchanged manually by exchanging emails with the corresponding files attached. Although some companies use encryption servers internally, these solutions often do not work across the companies boundaries.

2.2 OpenEngSB

The basic architecture of the OpenEngSB has been outlined in section 1.3. In this section the components relevant for the security implementation are described in more detail.

Locations

A workflow in the OpenEngSB consists of two parts: the abstract workflow description and the wiring description. In abstract workflow description, actions are annotated by using domain placeholders. In order to execute the workflow, these placeholders must be replaced by the actual connector instances. A wiring description is valid for a specific context. It holds information on which connector instances to use. Connector instances are provided as services that can be associated with arbitrary properties. The properties starting with "location" are used to assign the connector instance a name in a given context. The workflow description uses the same names, so that the descriptions can be combined when starting a workflow. Example: A workflow is executed in the context "testproject" and a node in the workflow uses a domain placeholder named "log". When the workflow is started this service is resolved by querying the service registry for a service with the property "location.testproject" containing "log".

Virtual Connectors

Virtual connectors are special types of connectors that do not implement any functionality. They usually delegate execution to other connectors depending on the usecase. In a workflow, a virtual connector instance appears as a valid value for domain placeholders. There are currently two types of virtual connectors: remote connectors and composite connectors.

[2] http://office.microsoft.com/excel/

[3] http://subversion.tigris.org/

Remote connectors when invoked, forward the invocation to a remote connector implementation on the network. They use the "ports" components to realize the communication.

Composite connectors are used to compose several connectors to one. A composition strategy decides how an invocation should be forwarded to the member connectors.

To other components, virtual connectors appears as a regular connector, making the distribution and composition transparent.

User Interface Architecture

The user interface is implemented using Apache Wicket [4]. All components of the user interface (pages, panels, etc.) are represented as Java classes. Instantiating these components is managed by the wicket framework. This way support for dependency injection and authorization control is added.

The user interface is divided in an abstract and a concrete part. The abstract part is a collection of reusable components that can be integrated in other user interfaces. The concrete part uses the abstract parts to provide an administration application.

2.3 Security in SOA

According to Hafner et al [6] in order to ensure interoperability in inter-organizational workflows, the partners need to establish a common understanding along three strands:

Technical foundation Although SOA are independent of programming languages or runtime environment, communication is still performed over interfaces. Common message formats and transport protocols need to be agreed on. This problem can often be solved by using established standards approved by OASIS[5] or W3C[6]

Global Workflow All partners of the network need to agree on which services to offer to create a contract the systems can be built on. This is a rather practical task and has to be solved for each usecase individually.

Security concerns All stakeholders may have their own security concerns which need to be taken care of. In distributed environments the lack of a central authority makes security a non-trivial task. The security policies need to be enforced on heterogeneous systems. Established standards like WS-Security, WS-Security Policy, WS-Trust provide some guidelines for to address these concerns.

Bertino et al [6] claim that the goal of securing web services can be decomposed into the following areas:

1. Integrity and confidentiality of messages

[4] http://wicket.apache.org/

[5] http://www.oasis-open.org/

[6] http://www.w3.org/

2. Enforcing of policies associated with a service

3. Authenticity of metadata for discovering and using services.

In order to provide security one must start at network level. Through network-level security integrity, confidentiality and reliability can be addressed. Bertino et al also state that "network-level security is not enough". Web service must be protected against unauthorized access. Therefore means for verifying a party's identity are required.

WS-Security

WS-security describes how to introduce confidentiality and integrity to SOAP messaging. It provides a mechanisms to associate "security-tokens" with a message. The standard does not enforce any specific types of security tokens, But provides a format to specify token profiles. Security tokens may be used for authentication. The standard defines "Claims" as "a declaration made by an entity (e.g. name, identity, key, group, privilege, capability, etc)" [45]. Security tokens represents a collections of such claims.

As the term "claim" suggests, they are not necessarily a prove of identity. Therefore the standard introduces "signed security tokens". These tokens are asserted by an external authority. This authority can endorse the claim by signing or encrypting the token. Examples for such signatures are X.509 certificates or Kerberos tickets.

The Message Security Model of WS-security uses signatures to verify the origin and integrity of the message. However the model is vulnerable to replay and man-in-the-middle-attacks because digital signatures are not sufficient for proper message authentication [45]. The standard strongly recommends to use include elements in the message to detect replays. Timestamps, sequence numbers, expirations and message correlation are mentioned as typical approaches.

WS-Trust

According to the process defined in WS-Trust a web service can require proof of claims. Requests are authorized based on transport security and the proofs attached to the message. One way for the client to prove its identity is to sign the security token with his private key. If the requestor does not have the necessary proof, it can contact other web services, that can provide the required proofs. This mechanism can be used to employ a Kerberos-like workflow. Services may only accept Tickets as proper proof of claims. The requestor must contact an authentication server, that authenticates it using a shared secret. The ticket provided by the authentication server can then be used to authenticate to other services.

WS-SecureConversation

The WS-SecureConversation [44] specification is based on WS-Security and WS-Trust and provides secure communication across one or more messages. WS-SecureConversation focuses on establishing a security context in which to exchange multiple messages. The security context is represented by a security token. Establishing the security context can be delegated to a

dedicated Security Token Service. When a service requests a new security context, the security token service distributes the context to other communicating parties. It is also possible for one of the communicating parties to initiate the security context. It creates a new security context and transmits the security token to the other parties by using mechanisms described in WS-Trust.

Extensible Access Control Markup Language (XACML)

XACML is an XML-based language used for describing access control policies. It is an OASIS standard available in version 2.0 [43]. In policy-elements the policies are described that serve as basis for access control decisions. A policy consists of one or more rules and a combining algorithm. Rule-elements contain boolean expressions that can be evaluated in isolation to derive an access control decision. Policies can also be combined into a policy set, which consists of several other policies or policy sets and a combining algorithm.

XACML specifies four combining algorithms for rules and policies:

- Deny-overrides: if a single element evaluates to "Deny", access is denied.
- Permit-overrides: if a single element evaluates to "Permit" access is granted.
- First-applicable: returns the result of the first element whose target is applicable to the request.
- Only-one-applicable: Works similar to First-applicable, but returns "Indeterminate" if more than one policy is applicable.

It is also possible to implement custom combining algorithms.

XACML also provides support for attributes based access control. It is possible to specify attributes on subjects as well es resources (objects). Attributes are identified using URNs and may contain multiple values. An example of a XACML-policy is shown in Figure 2.4.

Through the use of attributes in policies, it is possible to implement RBAC using XACML though a profile specification without modifications to the XACML standard [1].

There also is a Web service standard describing how to implement access control in webservices using XACML [2]

Ferraiolo et al point out that XACML "does not deal with all types of objects" and "is not comprehensive" and propose the development of a more abstract meta model for access control [23].

Security Assertion Markup Language (SAML)

The Security Assertion Markup Language is used to exchange Authentication and Authorization information in heterogeneous security infrastructures. It defines three kinds of statements:

- Authentication: contains information about how and when the subject was authenticated
- Attribute: contains attributes of the subject
- Authorization Decision: contains the result of an access control decision

```
<?xml version="1.0" encoding="UTF-8"?>
<Policy xmlns="urn:oasis:names:tc:xacml:2.0:policy:schema:os"
  PolicyId="urn:oasis:names:tc:example:SimplePolicy1"
  RuleCombiningAlgId="identifier:rule-combining-algorithm:deny-overrides">
  <Description>
    Med Example Corp access control policy
  </Description>
  <Target/>
  <Rule RuleId="urn:oasis:names:tc:xacml:2.0:example:SimpleRule1" Effect="Permit">
    <Description>
      Any subject with an e-mail name in the med.example.com domain
      can perform any action on any resource.
    </Description>
    <Target>
      <Subjects>
        <Subject>
          <SubjectMatch MatchId="urn:oasis:names:tc:xacml:1.0:function:rfc822Name-match">
              <AttributeValue DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name">
                med.example.com
              </AttributeValue>
              <SubjectAttributeDesignator
                AttributeId="urn:oasis:names:tc:xacml:1.0:subject:subject-id"
                DataType="urn:oasis:names:tc:xacml:1.0:data-type:rfc822Name"/>
          </SubjectMatch>
        </Subject>
      </Subjects>
    </Target>
  </Rule>
</Policy>
```

Figure 2.4: Example of an XACML policy using a subject attribute

A SAML authority is responsible for issuing assertions, that can be used by other systems to make access control decisions.[54]. The exchange of these assertions is used to achieve single sign on in heterogeneous environments. The advantage of SAML is, that the user only needs to supply the authentication credentials once. Other systems and domains rely on the assertions provided by the SAML authority. [21].

Both SAML and XACML each provide profiles to enable interoperation with each other. [4, 3]. The XACML Attribute Profile in the SAML profile specification describes the proper method for mapping SAML and XACML attributes, so that the SAML attributes can be used as input to XACML authorization decisions. The SAML profile of XACML specifies methods for using XACML queries in SAML requests.

2.4 Common Threats and Vulnerabilities

In RFC 1704 [32] some categories of attacks on authentication protocols are defined:

- Active Attack. In active attacks the content of the packages is modified. This is done by intercepting a package and modifying or by inserting a new package in the stream.
- Passive Attack. The data stream is passively monitored but not modified.
- Replay Attack: In this type of attack a message is recorded. The goal is to reuse part of the message to forge new messages that appear to be authentic.

Hafner et al [31] state that attacks on SOA based systems may be based on the same vulnerabilities as for traditional systems. Additional threats and vulnerabilities are described:

- XML specific attacks These attacks are targeted directly at the XML processing components like for example "XML-Bombs" (XML documents with endless recursions). This threat can be countered by using a validation service as security proxy.
- Service Scanning The target is profiled to identify potential weaknesses. In SOA public information stored in the WSDL files can be used to analyze a service.
- Compromised Services In environments with distributed services, information is required from a service repository. If the information in the repository is compromised, all services that look up information can obtain the manipulated data. This threat can be countered by authenticating each service provider.

Hafner et al also mention a family of replay attacks. They are based on the fact that web services are generally stateless. This means that messages can be recorded and replayed at any later point in time. To counter these replay attacks, messages must be authenticated through digital signatures and timestamps.

2.5 Authentication Mechanisms

There is a wide range of authentication mechanisms available. While simple mechanisms use only a password to authenticate a user, more complex credential types are also in use. When constructing a generic authentication system, these complex authentication types must be considered.

Public Key Authentication

Using asymmetric encryption can greatly improve security in authentication processes. These mechanisms rely on a previously shared key-pair consisting of one public key and one secret private key. While the public key can safely be publicly exposed, the private key remains secret to its owner. Authentication mechanisms only require the authentication authority to know the clients public key [52, 18]. Authentication mechanisms use signatures and encryption to ensure achieve their goal. The latter can also be used to provide confidentiality protection.

The downside of public key infrastructures is, they are hard to maintain and rely on certificate authorities to verify the authenticity of a public key. Schneier et al [20] state there are many security risks involved in the initial authentication process as well as in the verification process. Also the trust in the certificate authorities is an issue.

One time passwords

To avoid password theft through eavesdropping attacks or other threats introduced by replay attacks, these systems use passwords that are only valid for one request. In RFC 2289 a one-time password system is defined. The mechanism "uses a secret pass-phrase to generate a sequence

of one-time (single use) passwords" [33]. So a secret passphrase must be exchanged securely before one time password authentication is possible. In order to generate a one time password this a secure hash function (i.e. not invertable) is applied to the secret passphrase N times. The next password is generated similar, but this time the secure hash function is only applied N-1 times. The resulting number is then converted to human readable characters. An obvious downside of this mechanism is, that it only generates a limited number of passwords, before the sequence has to be reinitialized or secret must be changed. An alternative approach by Chang et al [14] eliminates this limit.

Multi-factor authentication

Several approaches exist where multiple authentication mechanisms are combined where each mechanism is used in one specific phase of the authentication process. Mizuno et al [41] present an implementation of multi factor authentication using multiple communication channels. The presented usecase uses the Internet and cellular mobile network as communication channels. An example presented works by sending One time passwords to the user via SMS message after the user transmitted his user id to the server by typing it into a web interface. This approach has to deal with the fact that SMS messages can be intercepted. However it requires the attacker to maintain to attack points in the environment. Another approach using biometric credentials was introduced by Bhargav-Spantzel et al [9]. One or more biometrics combined with other secrets like a password are combined to a multi-factor protocol.

2.6 Authentication Protocols

To actually implement an authentication process, a certain sequence of messages and events must be defined. For some of the mechanisms there are defined and well established standard protocols available.

HTTP authentication

RFC2617 [27] defines authentication mechanisms to be used in conjunction with the Hypertext transfer protocol [25]. The RFC defines two authentication methods: HTTP Basic Authentication and HTTP Digest Authentication. Both rely on a previously shared secret (i.e. password)

Basic Authentication

The basic authentication protocol defines that the user identification and the password are transmitted through the network in plain text. Because of the associated security issues it is suggested not to use basic authentication for security critical purposes. It should only be used in combination with other security mechanisms that are able to mitigate the original threats. Otherwise it is strongly recommended to replace it with the more secure HTTP Digest Authentication.

Digest Authentication

In HTTP Digest Authentication the password is no longer sent in plain text. Instead the server provides a “nonce”-value (“nonce means used only once”, [47]). The client calculates a checksum of username, password and the nonce-value and sends it to the server. The nonce is a “server-specified data string which should be uniquely generated each time a 401 response is made” [27]. The value of the nonce is defined by the implementation of the server. Digest Authentication is claimed to be not the most secure solution. Limitations include lack of confidentiality and vulnerability to replay and man-in-the-middle attacks. While replay attacks can be made very hard with appropriate nonce-implementations [27], the threat of man-in-the-middle attacks remains. Because of the fact, that this is not solvable with Digest Authentication, the use of more secure protocols (e.g. SHTTP, TLS) is suggested [27].

Secure Shell (SSH)

The SSH Authentication protocol [62] supports multiple authentication methods:

- password
- publickey
- hostbased
- none

SSH Authentication protocol assumes that all messages are transmitted using a protocol that provides integrity and confidentiality protection. It is possible to use SSH Authentication without an underlying protocol that provides confidentiality. Then however the standard recommends to disable the “password” authentication method [62].

Because the “none” method is not recommended for use and the “hostbased” method is rather inflexible, only the “publickey” method is considered here.

In order to initiate a “publickey”-authentication the client sends message signed with a private key to the server. The server who knows the users public key then verifies the signature. If the signature is valid the authentication is successful.

The message the client uses for generating the signature has the following variable contents:

- Session identifier provided by the lower-level protocol
- User name
- Service name
- Public Key Algorithm name
- Public Key

Further attributes are used to identify the message as "publickey" authentication-request [62]. The public key is included in the authentication request for performance reasons. The signing operation may involve expensive computation. Also a user may associate multiple valid public keys with his account. However the signature verification is only attempted with the key matching the supplied key. The signature is validated with the supplied key. A successful signature validation results in a successful authentication.

Kerberos

"Kerberos is a distributed authentication service that allows a process (a client) running on behalf of a principal (a user) to prove its identity to a verifier (an application server, or just server) without sending data across the network that might allow an attacker or the verifier to subsequently impersonate the principal." [48] The main intent of the protocol is authentication. Support for integrity and confidentiality protection is defined but optional. Neuman et al [48] describe ways for using Kerberos for authorization too.

Kerberos performs authentication by using conventional symmetric cryptography. So a user's secret key (password) are known to both, the user and the Kerberos server. The protocol is based on an authentication protocol proposed by Needham and Schroeder [47].

The Needham Schroeder Protocol

If *A* wants to establish communication with *B*, a Conversation Key (CK) must be obtained from an Authentication server. *A* sends a request to the authentication server containing its own name, *B*'s name and a "nonce" value in clear text. The authentication server calculates a new conversation key and sends a response to *A* containing:

- The initial "nonce value"
- Name of *B*
- The Conversation Key
- The Message *A* must send to *B*. It contains the Conversation Key and *A*'s name and is encrypted with with *B*'s secret key.

Only *B* can understand the encrypted message *A* received as part of the response from the authentication server. In order to prevent replay attacks additional handshakes are added to refresh conversations.

The Kerberos Authentication Protocol

In the Kerberos protocol, the Conversation Key is called "Session key". The part of the message the client (*A*) receives that is encrypted with the server's (*B*) secret is called "ticket". The ticket contains additional information:

- Client's name

- Server's name
- Client's network address
- timestamp
- expiration time
- Session key

The ticket is attached each time the client requests a service on the server so that the server can verify the information provided by the client. In addition to the ticket the client must include an authenticator in each request. An authenticator may only be used once. It is encrypted with the session key and contains the client's name and network address along with a timestamp. The use of these timestamps serves as an effective protection from replay attacks. For this to work all clocks are assumed to be sufficiently synchronized.

Kerberos defines tickets to be valid for only one single server. So when requiring services from multiple servers, the client must obtain tickets for each of the services. To ease this process a Ticket Granting Service is used. This service provides a method for obtaining tickets for other servers. Since this service is also part of the Kerberos environment, the client requires a ticket (Ticket Granting Ticket) to request the service. The ticket granting ticket can be obtained from the authentication server. With the ticket granting ticket, the ticket granting service can be used to get tickets for all other services. So the authentication process is done only once. Usually the ticket granting service and the authentication server are combined to a "Key Distribution Center" (KDC).

2.7 Authentication Delegation

In distributed environments and especially when tool integration is involved, it is often necessary for services to authenticate with another service to perform an operation on behalf of the invoking principal. An approach is to preserve the authentication using Kerberos. Other approaches [12] assume all entities in the system to support a security framework based on public key infrastructure that uses the Security Assertion Markup Language to determine authorized actions.

Credential Storage

A generic approach to automatically authenticate with third party system is to just store the credentials. Background task query this storage and authenticate automatically on behalf of the user. This technique is integrated in operating systems to enable secure storing of passwords for several services like email or browsing websites. MacOS ships with Keychain while Linux provides KDE Wallet[7] and Gnome Keyring [8]. Microsoft Windows provides an API calld "CryptProtectData"[9] for third party applications to store information that can only be decrypted on the

[7] http://utils.kde.org/projects/kwalletmanager/

[8] http://live.gnome.org/GnomeKeyring

[9] http://msdn.microsoft.com/en-us/library/aa380261.aspx

same machine by the same user. All approaches use the password the user provides at login to decrypt a password store for third party applications to access. These match the approach Schneier et al [22] describe as "Human Memory", because the actual secret is a password to be remembered by a human. All passwords are encrypted with one randomly generated master key. the master key itself is encrypted with a password. Alternative credentials to such a credentials store are biometric data or hardware tokens with displays to enter a password.

In distributed SOA environments the only possibility to provide such a store that makes the delegated authentication transparent to the user is to store the credentials on a server. Schneier et al refer to this method as "Single Sign-On". Every time credentials are needed the Authentication server is queried for the credentials instead of the local storage.

Other Single Sign-On (SSO) systems do not rely on actually retrieving credentials, but implement more complex key exchange protocols so that the actual password is never transmitted. These systems however require the target application to be compatible with the corresponding Single Sign-On system. Integrating SSO into a legacy application can introduce compatibility problems. Fleury et al claim that "in practice, implementing SSO with such services often involves storing the user's password locally to re-authenticate the user" [26].

Kerberos is considered the first such system [13]. More recent single sign-on implementations use SAML tokens to transport generic identity proofs [5]. SAML is part of several SSO implementations like the Liberty Alliance project[10] and the Shibboleth Project[11].

Another approach to SSO is "OpenID Authentiation" [12]. It is intended as a Web SSO and "targeted for Internet-scale adoptions" [56]. The protocol relies on the exchange of messages via HTTP or HTTPS. When the user wants to proof its identity to some web applications that acts as an OpenID "Relying Party", it forwards an identifier provided by the user to an OpenID Provider. The provider authenticates the claimed identifier and signs the response [50].

The OAuth protocol as defined in RFC5849 [34] enables third parties to access certain resources on behalf of the resource owner. These third parties are authorized by the user without providing them with the original password. Although OAuth focuses on access control it can be used as an SSO system too. Social networks like Facebook [13] and Twitter [14] use OAuth in the APIs for third party applications. Implementations of OpenEngSB connectors for Facebook and Twitter use OAuth for authorization on the remote account.

2.8 Channel Security

Securing the communication channel serves mainly the purpose of providing confidentiality protection. Messages exchanged over a secure channel can be intercepted but the content can not be discovered by a third party that is not part of the agreed session. Channel security is generally achieved by using symmetric or asymmetric cryptography.

[10] http://www.projectliberty.org/
[11] http://shibboleth.internet2.edu/
[12] https://openid.net/specs/openid-authentication-2_0.html
[13] https://facebook.com
[14] https://twitter.com

Transport Layer Security (TLS)

One of the most popular protocols to provide channel security is the Secure Sockets Layer Protocol (SSL). The Transport Layer Security (TLS) protocol is based on SSL and an IETF standards track protocol [17]. The protocol consists of two layers: the Handshake Protocol and the Record Protocol.

The TLS Record Protocol ensures that a connection is private and reliable. Symmetric cryptography is used to encrypt all data sent across the connection. The key used for encryption is only valid for single connection. It is negotiated by another protocol. To achieve integrity the protocol includes an integrity check using a keyed Message Authentication Code (MAC).

The TLS Handshake Protocol provides means of authentication and secure negotiation of a session key. Identities are authenticated using public key cryptography. The negotiation of the session key is both confidential and reliable meaning an attacker is not able to obtain the session key by eavesdropping or modifying messages in the communication. Although the TLS Record Protocol is intended to be used with the Handshake Protocol, it can be used in combination with any other negotiation protocol.

SSL and TLS are very widely used protocols in the web. They are used for secure communication with web applications and email servers. In fact the use of HTTP Basic Authentication is quite common, since it is often combined with SSL for channel security. As soon as a secure connection channel is established it is safe to transmit the password directly and unencrypted to the server. This combination is also a widely used practice for web services [46].

Message Encryption

SSL and TLS work on the transport layer of the communication. Nakamur et al describe two major problems with using those protocols in the context of web services [46].

- SOAP messages are not necessarily transmitted directly, but may include intermediaries.
- With SSL/TLS it is not possible to encrypt only parts of a message.

It is suggested to encrypt the SOAP-message using the XML Encryption specification [19]. XML Encryption works with arbitrary encryption algorithms and supports symmetric as well as asymmetric ones.

2.9 Access Control

Access control is used “to limit the actions or operations that a legitimate user of a computer system can perform” [53] Many previous approaches to access control focus on controlling access to data in a database system. With the increasing importance of web services it became inherent that specific access control strategies are required [6].

In a database management systems common access control mechanisms are invoked whenever a subject tries to access a data object. The rights of the user are checked against a set of authorizations. In an authorization the actions the user is authorized to perform on certain objects is described [8].

Bertino et al [6] define access control as a model that "restricts the set of service requestors that can invoke Web service's operations". Providing access control for services however presents additional challenges. Web services are considered stateless in most approaches. In stateful web services the state of the service has to be taken into account when performing access control decisions.

Also web services can be composed. The composition is presented as a web service as well, thus rendering the composition transparent to the service requestor. When several component services are combined into one composite service additional issues related to access control arise.

The component web services may have their own access control policies in place. One solution approach is to combine all policies to create a proper policy for the composite service. Another approach is to define new policies for the composite service. These new policies do not necessarily take all component services' policies into account.

Also the policy enforcement can be performed centralized or decentralized. In a centralized approach the composite service performs the access control decision for all component services using a policies derived as described above. If the composite service's policy takes all components into account, it also may involve component services that are not actually required in a particular invocation. In a decentralized approach each service makes a decision based on its own local properties. This could lead to necessary rollbacks if only some of the component services deny access.

A common example for a composition is web service orchestration. Orchestration is used to combine web services in a workflow. Often the Web Services Business Process Execution Language (WS-BPEL) is used to describe the workflow and the services used in it. The workflow itself is then registered as a service as well.

Access Control Models

An access control model is a defined set of criteria to design policies. System administrators can use these to define users' permissions.

The following models are considered the classical access control models: Discretionary Access Control (DAC), Mandatory Access Control (MAC) and Role-based Access Control (RBAC) [7, 53]

Discretionary Access Control (DAC) Access control decisions are based on the user's identity and authorizations. In this context authorizations stand for any means of describing rules for each user and each object whether the user is granted access. Each request is checked against the specified authorizations. If one of the authorizations states that the user is allowed to access the object, access is granted, otherwise it is denied. Inverted approaches are also possible where the authorizations hold information about what objects are forbidden for the user. Access is then only granted if no authorization states otherwise. There are also ways of combining positive and negative authorizations [53].

Mandatory Access Control (MAC) All users and objects are assigned security levels. The security level is supposed to reflect the sensitivity of an object. Access control decisions are

based on the relationship between the security levels of the user and the object. Depending on the type of access the relationship must satisfy the "read down" or the "write up" principle. "Read down" means that "a subject's clearance must dominate the security level of the object being read". "Write up" means that "a subject's clearance must be dominated by the security level of the object being written" [53]. This system may be extended by using categories to introduce finer grained security classifications. For a positive access control decision the user's categories and the object's categories must have at least one common element. [53]

Role-based Access Control (RBAC) Access control decisions are based on the activities the users execute in the system. "A role can be defined as a set of actions and responsibilities associated with a particular working activity" [53]. So authorizations are specified on roles instead of specific users. Users are granted permissions to adopt roles. For each action the user has to play a role in which the action should be executed. This way, designing a role and assigning the permission to adopt the role can be separated in two tasks, executable by different individuals [53]. A specification for a NIST standard for RBAC was introduced by Ferraiolo et al [24].

Roles are also a good way of mapping a organizational hierarchy into a tree of roles (hierarchical roles) [53]. The NIST standard [24] defines a seniority relation between roles: "senior roles acquire the permissions of their juniors, and junior roles acquire the user membership of their seniors" [24]. There are two types of role hierarchies:

- General Hierarchical RBAC: A arbitrary partial order can be used to represent the hierarchy.
- Limited Hierarchical RBAC: There are restrictions on the structure of the hierarchy (most commonly limiting it to trees)

Trees are a widely used format for describing role hierarchies [35].

Attribute-Based Access control

Attribute based Access control can be viewed as a generalization of other access control models. Permissions can be defined on any security-relevant property, known as attribute [63]. Attributes can be attached to subjects as well as objects. Yuan et al also point out that environment attributes (current date, security level, ...) while often ignored may also be an important part of an access control decision. The XACML specification is based on attributes and thus a suitable representation of ABAC.

2.10 State of the Art

The state of the art for secure communication in service oriented environments is to rely on defined standards as defined by Oasis (WS-Security) or W3C (XML-Encryption). Most standards provide protocol specifications based on XML-dialects and assume message exchange to

be performed using SOAP messages and HTTP. They also contain security guidelines that can be applied to environments relying on other protocols. The standards are defined to be independent of underlying encryption algorithms. Current implementations rely on algorithms like AES [16] and RSA [52]. The standards and algorithms are assumed to be secure. Therefore the work done in this thesis makes use of these standards as much as possible.

CHAPTER 3

Research Questions

Previous solution approaches for integrating tools in heterogeneous engineering focus greatly on integrating tools and their data. The concept of the "Automation Service Bus" (ASB) described by Biffl et al [10] focuses on integrating tools and their data while omitting security concerns.

This thesis is related to the security research for Technical Integration and Security in the context of Automation systems engineering [1]. Figure 3.1 shows the planned research in the Christian Doppler Laboratory for Software Engineering Integration for Flexible Automation Systems. Each number in the Figure corresponds to a research issue identified for future research:

1. The Engineering Service Bus Platform
2. Engineering Workflows: method for specification, design, and validation
3. Secure Engineering Workflow: method for specification, design, and validation
4. A Security Broker for EngSB environment

This thesis focuses on the design and implementation of the security broker which serves as a foundation for Secure Engineering Workflows.

In this theses the focus will be the design and implementation of the security broker and the interfaces to the components directly connected to the security broker (Security KB and User Management). Also the integration of the Security Broker on the gateway layer will be discussed. An integration solution for secure communication with external entities (e.g. other EngSB).

The main challenges are illustrated in Figure 3.2. One is establishing a secure communication between two parties. The figure shows an engineer using a tool accessing resources in the OpenEngSB over a network connection. Actions in the OpenEngSB involve triggering workflows. Workflows are defined independent of specific tools. So the access control performed

[1] http://cdl.ifs.tuwien.ac.at

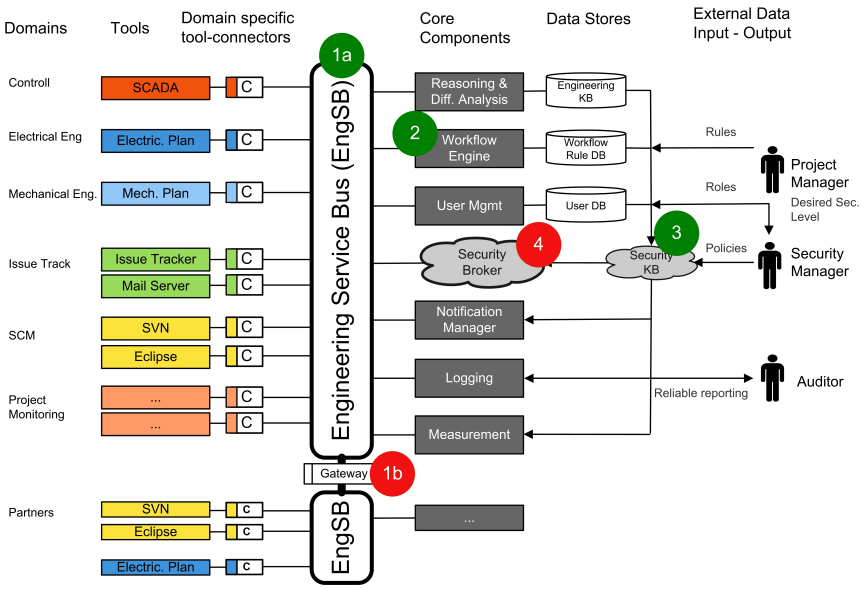

Figure 3.1: Planned research on technical integration and security [11]

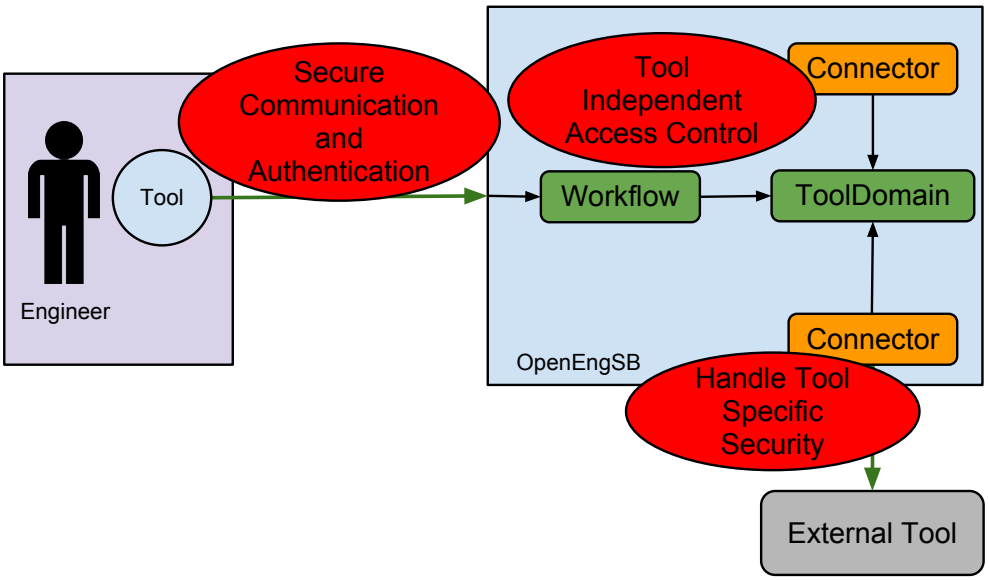

Figure 3.2: Challenges Overview

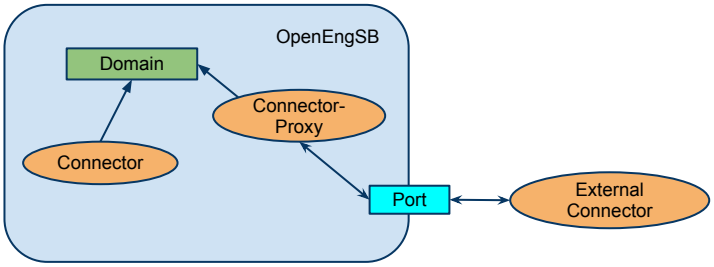

Figure 3.3: Difference between internal and external connectors

by the "Security Broker" component (see Figure 3.1) must be tool independent. However some tools connected to the OpenEngSB through tool connectors provide security features of their own that also must be handled during integration.

The assets that are to be protected are infrastructure resources represented as services and all kinds of data exchanged and manged with these services. The goal is to allow secure integration of data and services and thus protect all assents from exploitation by other parties. Exploitation means any unauthorized access or modification of artifacts (e.g. an electrical plan of another developer team) or triggering unauthorized infrastructure actions (e.g. running a simulation on the company's server cluster). For this to work all participating parties in the integrated environment require some common method of authenticating each other. Once authenticity has been verified each party can make access control decisions based on the provided identity.

In the OpenEngSB these interactions are always mapped to service requests. Data is also managed and controlled by services. There are two ways of using these services:

User Interface The OpenEngSB provides a web-application based user interface that can be extended in client projects. The web-application is able to directly obtain the services from the registry.

Remote Exchanging messages over the network using some arbitrary protocol. The service request is parsed from the message and then forwarded to the defined service.

Both variants must rely on the same data for authenticating and making authorization decisions. However they do not necessarily use the same mechanisms.

The OpenEngSB aims to remain independent of protocols and specific external applications so that the underlying protocol and applications can easily be exchanged and integrated. Because of the extensible architecture it is also important to enable security for third party extensions. Third party extensions are external connectors or services that do not run inside the OpenEngSB itself, but in some other environment (see Figure 3.3). Such extensions may be other programs running on the same machine or client-applications on remote machines on the network. In general there are two kinds of remote clients:

External connectors They provide an implementation of an existing domain in the OpenEngSB. For other services inside the OpenEngSB, they appear like regular connectors and they are integrated into workflows as such.

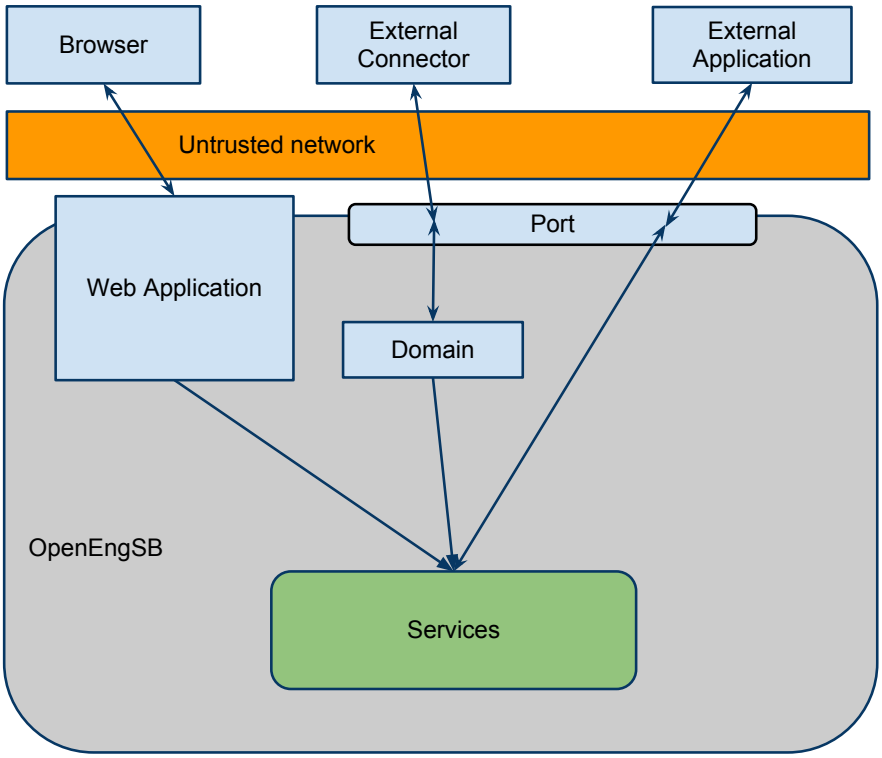

Figure 3.4: How external entities interface with the OpenEngSB

Remote collaboration applications External applications that explicitly exchange events with the OpenEngSB. The OpenEngSB itself is such an application. It uses these remote events to integrate workflows across company boundaries.

In both cases a separate communication layer is used to exchange messages with these external systems. The OpenEngSB does not enforce any specific protocol or data format for this communication. It uses an extensible plugin system so that support for more protocols can be added at any time.

Figure 3.4 shows where communication over a potentially untrusted network is happening and how it is handled in the OpenEngSB environment. A Web application may use the services as part of its business logic. External Connectors and Applications communicate through a exchangeable ports, where ports provide a specific protocol and format for communication. Protocols used for these ports are not required to support any security features. So it must be ensured that all security aspects are always automatically covered by any port implementation.

The SOA-based architecture highly suggests performing access control at the service level. This means that security assertions (authentication, permissions) are checked right before the service-methods are called. The service remains easily reusable across the system and remote hosts in a secure way.

3.1 Generic Authentication via Arbitrary Protocols

Authentication is a rather easy task in web applications. There are standards and frameworks supporting secure authentication via HTTPS. However some protocols that may be used for external third party extensions may lack of any mechanisms for providing sessions, authentication or channel security. So a highly portable authentication mechanism has to be established. All communication should be secure even when all traffic is assumed to be transported over potentially insecure or untrusted networks (e.g the Internet). Among other threats it has to be ensured that it is not possible to intercept authentication credentials (e.g. passwords) or to impersonate another remote client. Because of the variety of tools that are integrated in an engineering environment, where each tool and connector may rely on a specific protocol for communication, it is important that the concept of authentication is independent of a specific protocol. Also the method of authentication is often dictated by company policies or availability of resources. While large enterprises may rely on Kerberos-based solutions, small enterprises may find it sufficient to use less sophisticated mechanisms to authenticate.

> RQ1: Authentication - Which methods/approaches allow authenticating parties in a distributed workflow with integrated tools in the EngSB context with the focus on minimal integration effort on the tool side?

How must such a common authentication model look like, so that the solution does not enforce any specific mechanisms or protocols but still is resistant to most common security threats? A great number of authentication mechanisms and concepts have already been designed and implemented in other contexts. The challenges with reusing them are identified and discussed. In the end the concepts are reused where appropriate to design a modular authentication manager that allows tool integration beyond boundaries imposed by incompatible authentication schemes. The resulting authentication protocol provides a secure authentication mechanism, where each part of the process remains exchangeable, to make it open for user- or company-specific adaptations. It is important that the solution is easily adaptable by organizations that decide to integrate their engineering environment using the openengsb.

3.2 Integration of Access Control into Domains

With authentication established, the target system can now make decisions based on the authenticated principle whether to grant access. But where exactly should the decision be made? As services are the assets an access control decision based on, the most likely choice seems to be somewhere right before invoking it. Among “core services” that are used for common task like executing most of the services are likely instances of tool connectors. Every tool connector provides functionality for at least one domain. Connector services should not be referenced directly but rather by the desired placeholder that represents a domain. For every project these placeholders may reference any connector instance (see Figure 3.5).

> RQ2: Which methods/approaches allow controlling access to infrastructure (services) and data (artifacts) with generic and sufficiently efficient mechanisms?

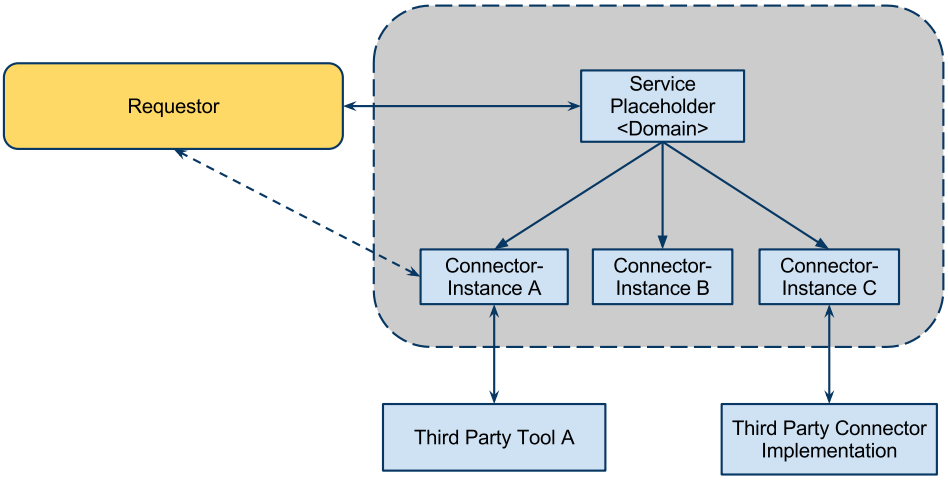

Figure 3.5: Illustration how the Domain abstracts whether an external or internal tool connector is requested.

The connector referenced by a placeholder can change any time. Because of the dynamic nature of these placeholders security must be independent of the underlying connector. So with the domain the only constant in this environment, security must be taken into consideration when designing domains.

Requirements and usecases of Access control will be analyzed to determine how certain interactions can be decomposed and represented as permissions. Also these permissions need to be assigned to security principles such as users. As a result of the flexibility of the authentication system, this assertion of these assignments also must be decomposed into similarly exchangeable components. Regarding the administration, the domain architecture will be used to allow exchangeable user- and role management components

Generic Access Control Model

The security assertions before the service calls require an underlying model to make decisions about granting access. A model for managing permissions and roles needs to be defined that supports these decisions. This model has to be very flexible and extensible, since some domains may introduce domain-specific security challenges. Access control requirements of domains may require interactions on a very high level of detail (e.g. control access of each line in a file). Also third party extensions should be able to fully integrate with the core user management system. However these third party extensions should only be able to influence the access control system in their own scope. Administration effort is a big concern here. If the solution is too generic and the administration too difficult the system might get used the wrong way giving too many permissions to users.

Administration

These access control data has to be maintained. Security policies must be translated into the provided models. The question of who is responsible for maintaining the access control becomes important. With increasing size of organizations this task may be too complex for a single person. What expertise is required to maintain these security policies? Managing permissions and roles in hierarchies may greatly reduce the administration effort.

It is likely that the responsibility for administrating the users and permissions, is divided among a dedicated "security manager" and project managers. In order to ease administration "Roles" can be used as an abstraction layer for responsibilities.

3.3 Integration of tools' security concepts

When integrating tools it must be considered that they might introduce their own security concepts that need to be integrated into the security system of the OpenEngSB. For example a lot of tools use a username and a password for authentication, but some use more complex systems like key pairs and ticket-based authentication. When integrating tools, these concepts have to be integrated too.

> RQ3: What additional challenges does the integration of tools with complex security concepts introduced and how could the be adressed?

A tool connector will be responsible for integrating any tool specific aspects. This means that authentication mechanism used for the tool is implemented in the corresponding tool connectors. However to actually perform any authentication some kind of credentials are needed. These credentials need to be available without user interaction, i.e. they need to be stored.

The result is a description of how the storage of credentials can be done safely using the mechanisms for authentication and authorization described before. In order to validate that this is sufficient, a description of how to apply the solution to three of the most common authentication mechanism types will be presented.

3.4 Impact of Security concept on OpenEngSB-Architecture

Adding security to distributed workflows and work environments has consequences for many stakeholders. Security is likely to increase cost for operating, administrating and maintaining the infrastructure.

The secure exchange of message may impose a certain amount of overhead in message size and quantity. The processing of encryption, authentication and authorization decisions is likely to increase workload on all clients and servers involved. But also the requirement of personal resources may change because of increased administration efforts.

Additionally there are consequences for developers creating new modules for the OpenEngSB as well. They will need to integrate new modules using the security mechanisms, making the overall development more complex. There are three main scopes of development, that are considered in this thesis:

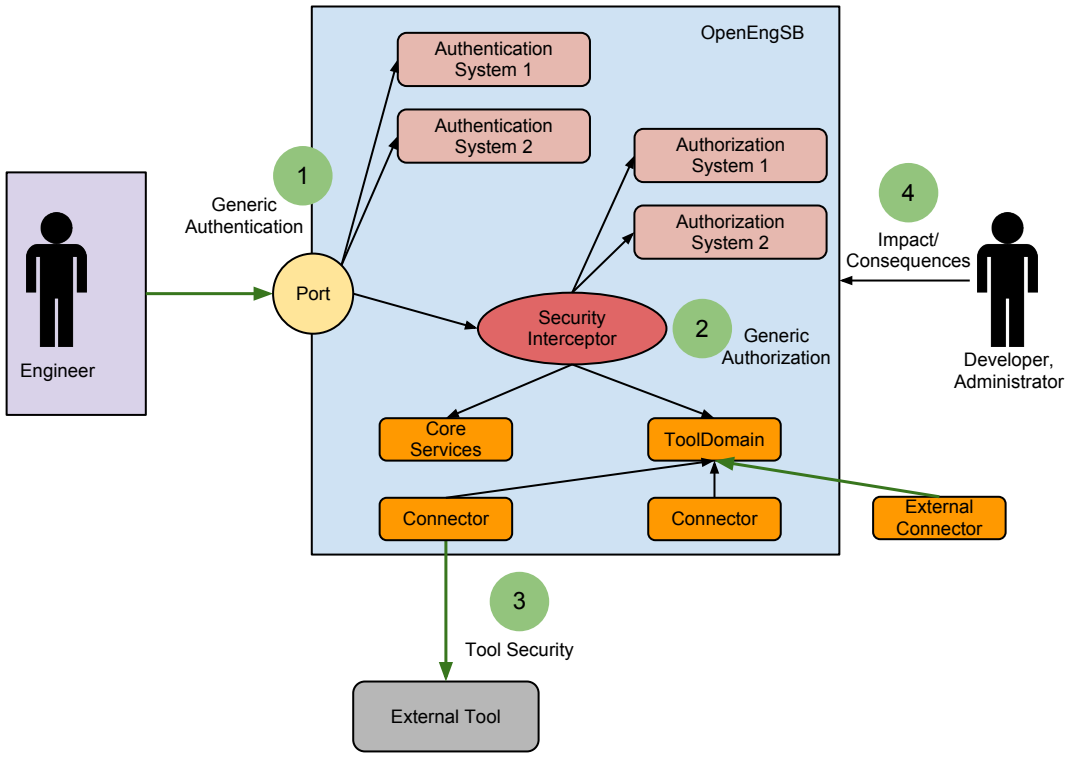

Figure 3.6: Research Question Overview

- Development of external components
- Development of projects using the OpenEngSB
- Development of the OpenEngSB itself

RQ4: What impact does the security concept and design have on the usefulness for customers and developers?

The result will be a compilation of possible cost increases resulting from the adaption of security.

3.5 Overview

Figure 3.6 shows an overview of all research questions and their context in the EngSB environment.

RQ1 Authentication - Which methods/approaches allow authenticating parties in a distributed workflow with integrated tools in the EngSB context with the focus on minimal integration effort on the tool side?

RQ2 Which methods/approaches allow controlling access to infrastructure (services) and data (artifacts) with generic and sufficiently efficient mechanisms?

RQ3 What impact do complex security concepts have on the design of tool integration? What are the design trade-off options to address the observed design impact?

RQ4 What impact does the security concept and design have on the usefulness for customers and developers?

The main contributions are illustrated in Figure 3.7. They include the following components:

- Secure way of transporting messages between two parties realized by "Secure Ports"
- A framework for integrating arbitrary authentication mechanisms
- A "Security Interceptor" enforcing access control policies on infrastructure and data
- Concepts for integrating tools with complex security in the OpenEngSB context

3.6 Research Approach

To derive solutions for the research questions above, a research method based on Constructive Research [37] was chosen. It involves the steps described below.

Identify Threats In order to get a clear definition of what is secure, potential threats have to be identified. The presented solution should mitigate these threats improving the overall security of the authentication (RQ1) and authorization (RQ2). Threats are identified by common practices and heuristics like the STRIDE approach.

Literature Research The identified threats are compared to similar threats described in literature. In addition generic work in context of technologies used in the OpenEngSB and concepts related to the OpenEngSB (e.g. ESB) is taken into account. The literature also includes standards that have been accepted by recognized standardization organizations as well as common practices suggested by commonly accepted but not binding standards (like RFC). Existing authentication mechanisms and protocols as well as authorization paradigms are analyzed on which concepts can be reused or directly integrated into the generic authentication mechanism (RQ1) and the authorization system (RQ2). More complex protocols and mechanisms which are used on some tools are also analyzed in order to understand some concepts that must be integrated (RQ3).

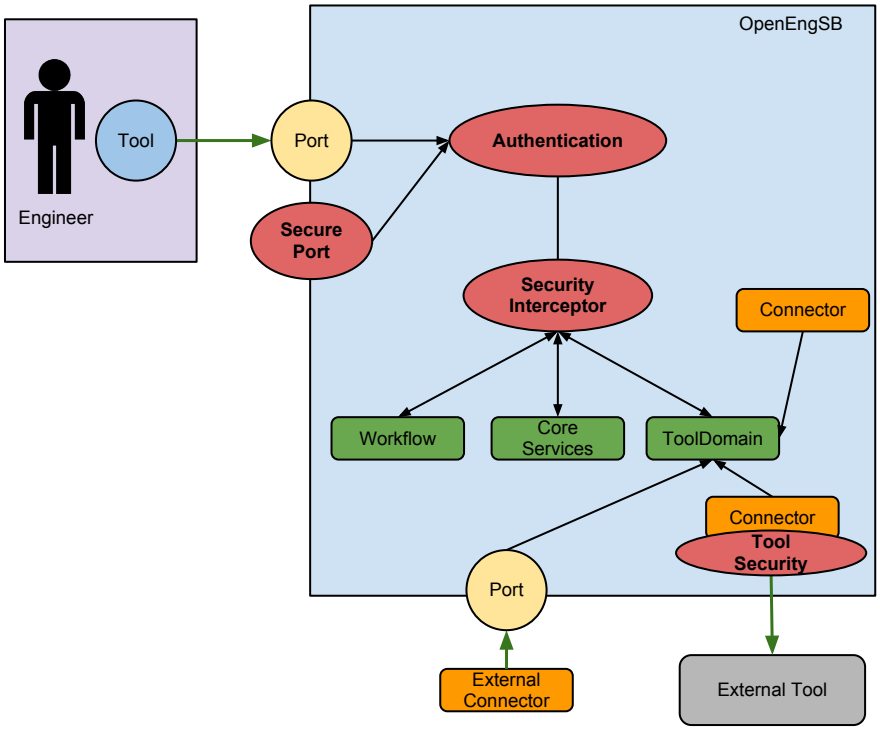

Figure 3.7: Contributions Overview

Construct usecases A set of usecases will help to illustrate the need and the impact of the security solution. Also they serve as means for validation of the presented security concept (RQ1, RQ2 and RQ3).

Derive solution concept Based on this theoretical work, models are created, and overall architecture is adapted to fit the security requirements. Components that serve as a framework to integrate authentication (RQ1) and authorization (RQ2) into domains and tool connectors and workflows are designed. Also some approaches for components that allow integration of tools with complex security are designed (RQ3).

Evaluate Frameworks When considering security there are a lot of common requirements for most of the systems. Authentication, Authorization and Confidentiality are part of most security solutions, so there is a number frameworks that provide secure abstract implementations. The goal is to choose a framework that provides as much already implemented functionality as possible. However it must also perform well in combination with other technologies used in the OpenEngSB. Also the framework must not limit design choices regarding the basic architectural concept of the OpenEngSB, i.e. it should in no way limit the possibilities of the OpenEngSB or bind the implementation to a specific tool.

Implementation After choosing the frameworks a prototype of the of the security concept is implemented on the OpenEngSB Open Source project. The prototype implements a framework for providing generic authentication (RQ1) and authorization (RQ2). After the implementation the impact on the architecture is analyzed to derive consequences for future development as well as administrating systems running the solution (RQ4).

Evaluation When the implementation is done the usecases are simulated through automated testing and staging real world scenarios. In a valid security solution, the described usecases must be realizable and secure (RQ1, RQ2, RQ3). A secure solution mitigates all priorly identified threats. This is done by simulating man-in-the-middle attacks as well as replay attacks with all along with all tested scenarios. An application for monitoring all network traffic is used to ensure that no sensitive data is transfered unencrypted. In addition some simple stress tests show the impact on the performance of the running system (RQ4).

CHAPTER 4

Use Cases

In this section typical usecases of the OpenEngSB are described with regard to securing the described interactions. These usecases are used to evaluate both the robustness and flexibility of the solution. The first usecase is a complex usecase covering many security aspects.

The usecases 2 and 3 describe rather isolated security aspects of the OpenEngSB with mostly technical challenges. In usecases 4 and 5 organizational structures are considered, and corresponding requirements are integrated into the security model. While usecase 3 focuses managing permissions and policies, usecase 4 targets the granularity of these permissions. In usecase 5 the security of the integration of external tools is addressed, since integrating a great variety of tools is one of the most important concepts of the OpenEngSB.

For each usecase an abstract description is given. Then an arbitrary real world scenario is designed, that is aimed to cover all challenges presented by the usecase. Based on the real world scenario a corresponding "test scenario" is designed. It serves as validation that the usecase can in fact be realized. At first integration tests are used for that. In integration tests everything that is supposed to run on a remote machine is started on the same machine (to ease automatic testing). The second validation step is to actually distribute the components on the network. This can be achieved using either physical or virtual machines. Because of the complex setup, this validation step must be executed manually.

4.1 Signal Exchange

A company is creating a complex electrical device. It requires several external devices plugged to a parent module. Some of these external devices are manufactured by an external partner company. To make sure the components are compatible the layout of the connections between the components are described as "signals". A signal is a common concept for all involved parties, but is represented differently by each software tool. The specification of signals is critical for proper interoperation of the components. When the signals change each participant in the project must be notified and the plans need to be changed accordingly.

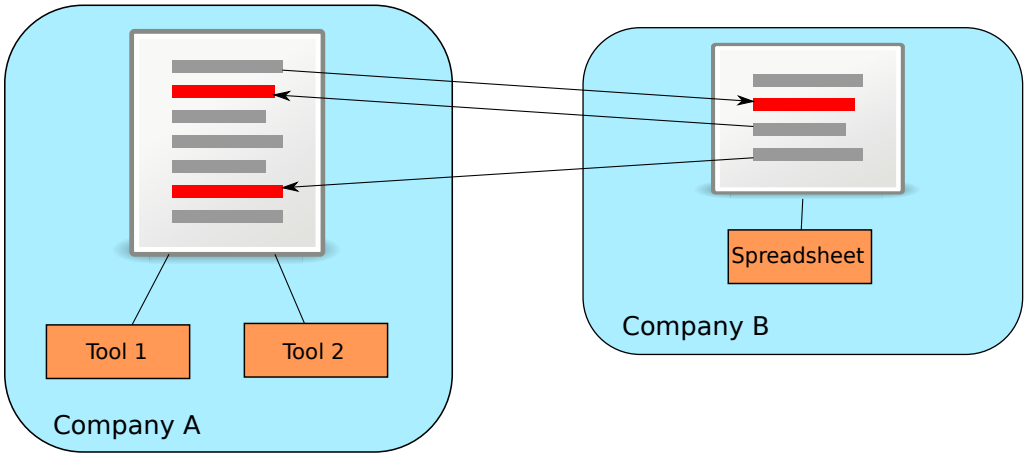

Figure 4.1: Two companies exchanging data independent of tools used

The signal changes are likely to be transported over the network to the other company. This means that each company shares parts of their artifacts with the other company (see Figure 4.1). The transmission must be secure. Also the companies want to protect other assets from exposure to the other company. So they must only be able to access the data, they are supposed to.

Scenario

Suppose Company A is working on the parent component. All electrical plans are maintained with a tool specifically designed for such plans.

Company B which is working on a module for the parent component. The part of the plan they require can be represented in tables. Therefore they maintain their plans using some spreadsheet software. Both plans already contain an initial specification of signals.

Now some employee in Company B makes changes in a spreadsheet. The tool is connected to Company A through an external tool-connector. As soon as the local copy of the signal specification of Company B is updated, the Company A gets notified of that change. The notification is transported as an Event that can trigger a predefined workflow (see Figure 4.2). Possible workflows could include notifying developers via email or creating tickets in an issue-tracker. In this scenario it is assumed that the changes are saved in company A's database and the changes are than propagated to all other tool-connectors associated with the project or artifact. Handling updated data is the responsibility of every tool connector. A possible implementation is to present the changes to the user in some kind of queue for manual merging. Connectors (internal or external) may only send Events when their local copy is up to date. The changeset contained in the event's payload is supposed to be applied on the agreed master copy.

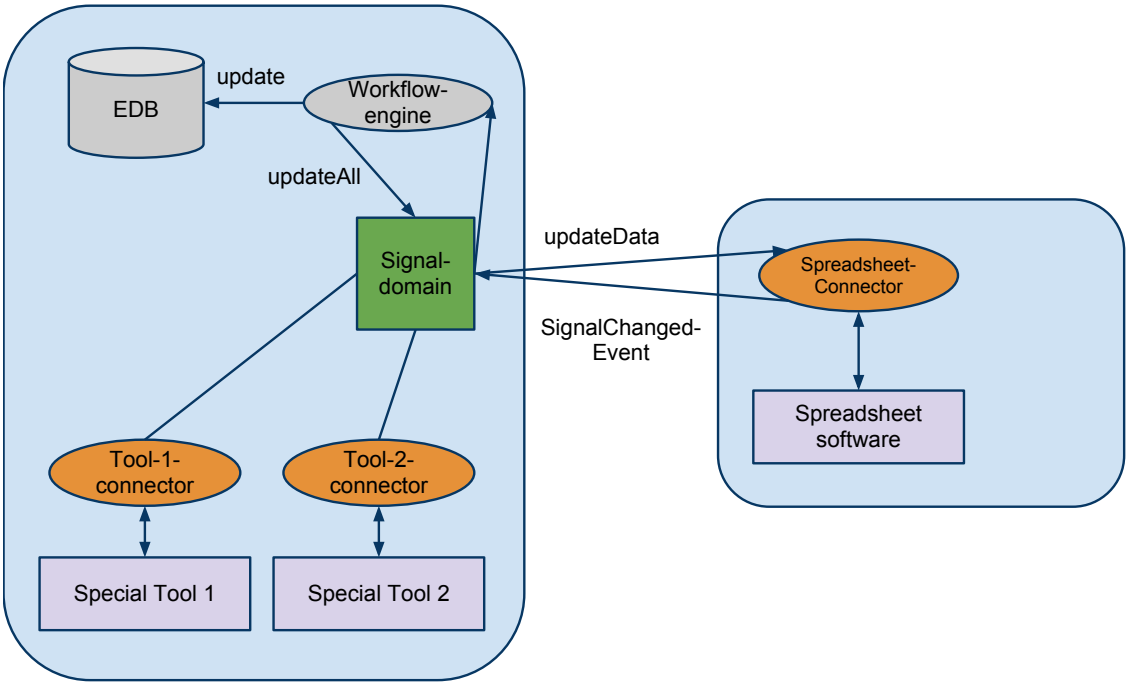

Figure 4.2: Customer Data Usercase architecture

4.2 Authentication of an External Connector

External connectors are connectors that do not run directly inside OpenEngSB-instances. The sender of the message must be authenticated in some way without relying on any mechanism in the underlying communication protocol (RQ1). They register themselves with an OpenEngSB-instance while maintaining the implementation elsewhere. The communication is done in a way, so that it remains transparent to other OpenEngSB components whether this connector is external or internal.
The OpenEngSB and the external connector exchange messages containing service calls and return values. These messages are exchanged both ways. It is possible to run external connectors on the same physical machine as well as some remote server in a local network or the Internet. Every action in the OpenEngSB is executed in the context of a specific user, that has predefined roles and permissions. This context must be synchronized to the external connector, to make sure that actions executed in the external connector are executed with the same permissions, as if it where an internal connector. Moreover when the external connector intends to invoke a service in the OpenEngSB it needs to authenticate the user to be granted access to the OpenEngSB.

Scenario

One aspect in the "Signal Exchange" usecase is, that the external connector used in Company B, must authenticate with the Bus in Company A. The external connector implementing the "Signal-Domain" uses Microsoft Excel tables for managing these variables. It is implemented in .NET and executed on a remote machine connected to the machine running OpenEngSB via an Internet connection. When the data in the spreadsheet is changed, the connector raises an event to be processed in the OpenEngSB. In order to invoke the security critical method "raiseEvent", the client must authenticate with the OpenEngSB. Both parties need to be sure that the remote application is not an imposter.

Test Scenario

An OpenEngSB is setup with an example domain installed. A external connector is installed on a remote machine. The external connector is a simple Java program that receives JMS messages and raises Events for each received message. This way the network communication scenarios are covered without relying on complex functionality in the connector itself. A proxy connector is setup to point to this external connector. A method on the connector is invoked and forwarded by the proxy. This causes the connector to log the execution and raise the corresponding Event. The OpenEngSB receives the event and adds it to a log file.

Now fake messages are injected on both sides. A forged method call request is sent to the external connector, and a forged Event is sent to the OpenEngSB. Both must reject the invocation.

In addition network traffic is monitored to verify that no credentials are transferred in plain text.

4.3 Hierarchical Policy Management

In the "signal exchange" usecase, it is likely that company A manages maintains several development teams to work on multiple projects. While controlling access inside the company is important by itself, it gains importance when collaborating with other companies and sharing resources. The first step towards introducing Access control into Domains (RQ2) is to design a model for managing permissions. To ease administration of the permissions roles are used. Access control policies often affect a certain scope (e.g. a project, a department, an employees position). These policies may reflect the organizational hierarchy in a company. These hierarchies get more complex with regard to project management, where employees may take roles that may differ from their role in other projects or the company itself. So users have authorities based various factors that are often subject to change like for example which projects they are working. These authorities have to be managed. It is important to specify which users get certain authorities, but also which users are allowed to grant these authorities.

Scenario

Company A operates two projects P1 and P2. In P1 they collaborate with Company B sharing all resources in the project. A developer that only works on P2 should not be allowed to access the signal-database in P1. A developer in Company B may only access signal-database in P1, but not P2. The CEO of company A is allowed to access any project's signal-database.

In this example there are at least two kinds of roles required:

Project specific To separate members of different projects

Management Global roles for administration across projects

Test Scenario

For validation the Source code management domain is used. It provides methods for accessing and modifying files in revision control software. The connector implementation for Git [1] will be used for the test. Each project uses its own SCM and a corresponding connector in the OpenEngSB. In this scenario it is assumed that each repository contains a file providing a description of a project. The domain's "get" method can be used to retrieve the current version of the file's content. Two projects are created and The necessary data is created:

- The users Alice and Bob
- The roles "CEO" and "engineer"
- Two projects "P1" and "P2"
- Two Git repositories containing a single file.
- Alice is assigned the role "engineer" in project "P1"
- Bob is assigned the role "CEO"
- Alice is assigned the role of "engineer" in the project "test-project"

When Alice tries to invoke the checkout-operation in the context of "P2", access must be denied. When Bob does it, it returns the document.

4.4 Fine granular Access Control

Restricting access to projects and users uses already existing concepts in the OpenEngSB. However these concepts only provide rather coarse grained abilities, to restrict access to important assets. It might be necessary to protect only specific parts of services, like:

- a subset of actions
- specific artifacts that are manipulated by the service

[1] http://git-scm.com/

Specific domains may even require more detailed control to implement domain-specific permissions. In the “signal exchange” usecase this can be important, when company A wants to restrict access to parts of the signals in the same project where they collaborate with company B.

Scenario

Company A operates a project where they collaborate with Company B. Company B provides one module required by company A to assemble the whole component. All other components are built by company A itself. All specifications on the connections with the modules are maintained in form of signal lists with the use of tools in the signal domain. Only a subset of the signals in the scope of the project are relevant for the collaboration with company B. Company A wants only to expose this subset, while restricting access to other signals.

Test Scenario

Like in the test scenario for Hierarchical Access Control, the SCM domain is used here. The main difference is that now all users work on the same project. In a project P1 a single SCM repository is used to store documents. The repository contains a directory named “internal”. The access to files in this directory is limited. The following data needs to be created:

- Two users “Alice” and “Bob”
- A Project “P1”
- A Git repository containing a directory named “internal” and some files
- A connector instance connecting the OpenEngSB to the Git repository.
- Alice gets assigned the permission to read all documents in the repository.
- Bob is only allowed to read data outside the “internal” directory.

Now when Bob tries to call “get” with a path that starts with “internal/”, access must be denied.

4.5 Continuous Integration

Executing and integrating predefined workflows in the OpenEngSB is a vital part of its functionality. Workflows may be triggered by external events or by user interactions. Parts of a workflow may also require human interaction. The user actions as well as the automatically triggered events should be executed with as few permissions as possible. Workflows are designed for a combination of domains, but are in general reusable with multiple combinations of connectors.

Scenario

In a project a variant of “Continuous integration and testing” (CIT) is used. When some developer commits changes to the projects SCM-repository an “SCMCommit” event is raised. This event automatically triggers the CIT workflow. In the workflow two domains “build” and “test” are called to verify the current version in the SCM-repository.

4.6 Tool Integration

Tools that are to be integrated using the OpenEngSB may already provide their own security features. These security-features must be integrated along with other aspects of the tool (RQ3). Some of the tools even require the use of some of these features (e.g. authentication in an issue tracker). So the OpenEngSB must be able to map authentication credentials and user permission of the domain to permissions in the tool’s security layer.

Test Scenario

Create a workflow according to the CIT model using the workflow engine provided by the OpenEngSB. Also some data needs to be created:

- a project “test-project”
- the users “Alice” and “Bob”
- Alice is assigned the role “project lead”
- Bob is assigned the role “developer”
- a remote Git repository
- according connectors with wiring for the CIT workflow

Also a continuous task watching for SCM-changes (to create the SCM-Commit-Events) is required. When someone commits changes to the SCM that workflow should start. When Bob authenticates and wants to invoke the workflow manually access is denied. When Alice authenticates she can start the workflow.

CHAPTER 5

Security Concept for a (Software+) Engineering Environment

In this section the concepts to implement a flexible and robust security solution for the OpenEngSB are described. At first the threats to OpenEngSB, its services and integrated tools are analyzed. Then the concept will be built with an iterative approach. With each iteration another research question described in chapter 3 is included in the considerations. In every iteration the threats are reassessed, to determine which of them can be mitigated.

5.1 Initial Threat Analyses

A common method in the context of threat modeling in a software system is the STRIDE approach. STRIDE is an acronym summarizing the following abstract threats:

- Spoofing of user identity
- Tampering
- Repudiation
- Information disclosure
- Denial of Service
- Elevation of privilege

Decompose to Components

The process requires the system to be decomposed to smaller components. The relevant components of the OpenEngSB are:

- Ports to Core Services

  Many core services of the OpenEngSB are exposed to the network to receive RPC requests. This is important, so that external systems interact with. Core Services include:

  - Workflow Engine
  - Context Management
  - Service Management

  This allows to fully administrate the OpenEngSB remotely.

- External Tool Connectors

  External Connectors are a special case of services. They are exposed to the network using the same mechanisms as services. The provided service serves as junction point between the OpenEngSB and some external program. These proxy services allow the OpenEngSB to treat such external connectors as if they were internal connectors. This means they are fully trusted, while the actual external application is not necessarily.

- Administration UI

  Usually services are used to provide actual business logic for user interfaces. However there are some special requirements for user interfaces (e.g. hide parts the user is not authorized to use) that need further integration with the security system. Also does the UI serve as a bridge to authentication and policy management. In this theses only the case of a web interface in a browser is considered.

Identifying Threats

When identifying the threats the network connection between two remote machines is assumed to be completely untrusted. This means that intruders can potentially interfere with communication in every way they want. In a worst case scenario they can intercept any communication before it reaches the target, and also can introduce any new message into the network. This means they can block, alter, copy, replay ongoing communication or even generate and send forged messages. [47] states that "while this may seem an extreme view, it is the only safe one when designing authentication protocols." A comparable scenario would be a client connecting to a server over the Internet using a wireless hotspot without any wireless security mechanism. With the popularity of unprotected hotspots provided in hotels, this scenario is not unlikely. When using such a connection, possibilities go from intercepting and inspecting all network traffic even up to man-in-the-middle-attacks.

For the initial threat analyses it is assumed that the current state of the system is without any means of security. All actions in the Administration interface and all service invocations can be done by anyone over the network. So for the initial threat analyses all components share the same threats. The solutions will of course differ.

Common Threats

- Spoofing of user identity

  With unprotected exposure of resources to the network, there is no way of telling who is currently interacting with the system. An authentication mechanism is required. It's important that an authentication can only be performed by the actual subject.

- Tampering

  Considering a man-in-the-middle attack where all data is intercepted and the attacker has complete control of the traffic. Data could be intercepted and altered before it reaches the server. Even if it is not possible to alter the data, it could still be abused to perform replay attacks. This can have potentially unpredictable consequences to the system.

- Repudiation

  With universal public access, there is no proof who performed a certain action. The auditing built into the workflow engine would have to be extended. While the presented concepts will provide some improvements in this context, it is not discussed explicitly.

- Information disclosure

  Consider all data sent across the network can be intercepted and stored for espionage purposes. It needs to be made sure, that this data is useless to external entities.

- Denial of Service

  Flooding the OpenEngSB with requests rendering the infrastructure unusable. This issue will not be explicitly discussed in this thesis

- Elevation of privilege

  It is possible for any user to perform administrative tasks (like creating new projects). A permission system needs to be established.

5.2 Authentication Domain

Depending on the required level of security a wide range of authentication mechanisms is used on companies today. Most companies authentication systems rely on assigning users one or more passwords to access the systems. More complex solutions are also used, like for example one-time passwords (either exclusively or as part of a two-phase-authentication). Another solution is using smartcards which contain more complex secrets like private keys for authentication mechanisms using asymmetric cryptography. Also A widely established system is using Kerberos for distributed authentication.

Every method provides a certain level of security. Some complex solutions requires complex client-side software, which makes it very hard to deploy and use in web-applications. So it's also possible that multiple mechanisms are used in the same company simultaneously. Also when providing external companies access to some resources, solutions requiring client-side software or physical devices are often unfeasible.

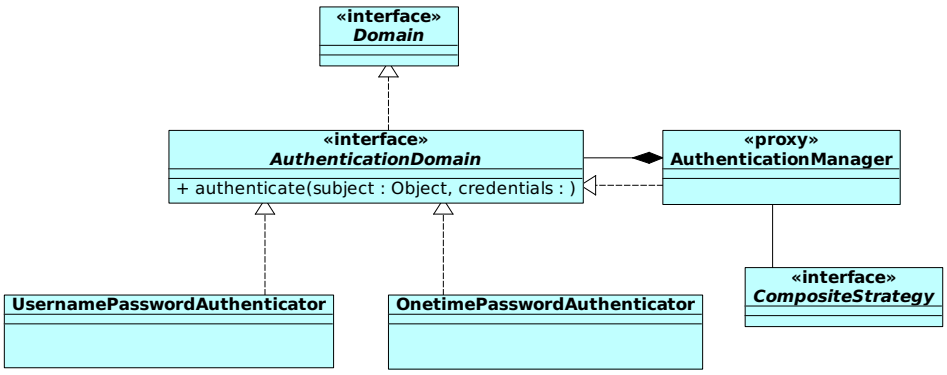

Figure 5.1: Authentication Domain with manager

So in order to support multiple Authentication mechanisms, the approach is to dynamically integrate authentication as connectors for the OpenEngSB. An "Authentication Domain" is defined. It serves as a common interface for developing plugins for authentication mechanisms. This solution provides great advantages, because it fits well in the concept and infrastructure of the OpenEngSB itself. The authentication process is delegated to authentication connectors (see Figure 5.1).

Implementing connectors for multiple authentication mechanisms is not sufficient. How should the connectors be discovered and used by other components? The discovery of connectors is done by assigning locations to connector-instances (see section 2.2). Authentication requires a special strategy for dealing with multiple connectors (see Figure 5.1).

A simple approach to such a strategy is, to iterate through all available connectors and try authenticating until the first one succeeds. If no connector can authenticate the user with the given credentials, the authentication as a whole fails.

Example: A company uses fingerprint sensors for authentication. As a fallback for some unsupported machines they also provide every user with a password. For simplicity it is assumed that both credentials provide the same authorities for each user. Two connectors need to be setup as illustrated in Figure 5.2. The shown "authentication strategy" shows the position of the strategy that is responsible for handling multiple connectors. When an authentication-requests comes in, the strategy queries the registry for all services matching the location "authentication/*" where "*" serves as a wildcard. The result of the query is supposed to be sorted according to some service ranking. In this case we assume most authentication-requests are done using fingerprints. So the Fingerprint-Connector is always called first. If it succeeds, a successful authentication is returned. If it fails, the Password-Connector is the next in the list and will be called. If authentication with this connector fails too, the authentication as a whole fails.

The same concepts can be used to provide a two-phase authentication mechanism. To reuse the previous example a slight modification is made. Suppose a successful authentication requires both, a correct password and the matching fingerprint. In this case only the strategy needs to be exchanged to always call both connectors and only return a successful authentication if both connectors successfully authenticate the same user.

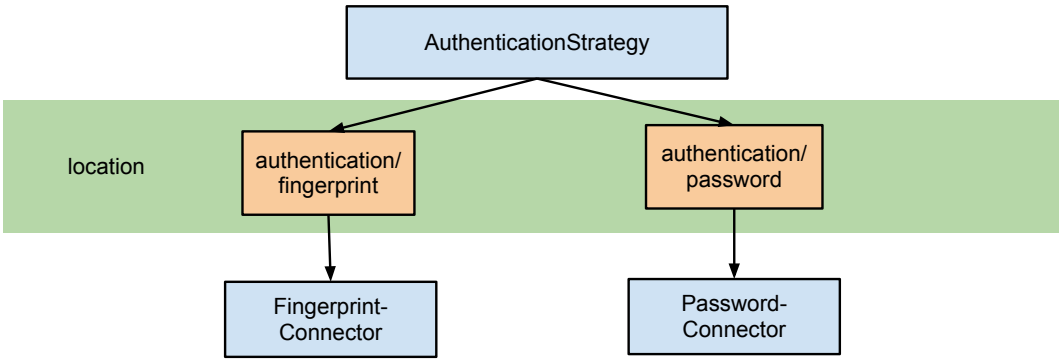

Figure 5.2: Authentication connector example

With authentication added to the infrastructure, spoofing an identity becomes more difficult. The authentication process is not secure yet, so it is still possible to intercept the authentication request and reuse it later.

5.3 Authentication Gateway

With an established way of implementing and using arbitrary authentication mechanisms, the next step is to use the authentication remotely. When authenticating over a network, the credentials must be transported securely. This section describes mechanisms for every scope of the OpenEngSB where authentication can be performed. These concepts described here, make it harder to spoof a user's identity, because the communication is secured against tampering and information disclosure by using encryption and integrity checks.

Administration UI

A common solution for authentication in web applications is requiring the user to input a username and password. It is assumed that the password was chosen and delivered to the user in a secure way. For authenticating a user in a web application there are a lot of predefined mechanisms and well established standards that help to achieve security in this context. Using HTTP Basic Authentication [27] transfers username and password to the server in plain text. This is problematic when the connection is established over an untrusted network. The traffic could be intercepted or monitored, and the credentials obtained by a third party who can use these credentials to spoof the users identity. The only viable solution described in RFC2617 is to use more secure variants of HTTP (like HTTPS [51]).

Ports to Core Services

While in web applications the protocol and the data format is predefined, this is not the case for the ports. A port may provide any communication protocol in conjunction with any data format.

A possible port implementation is using web services and XML SOAP as message format with HTTP as transport protocol. While using SSL might not always be possible because of the restriction to direct communication and two parties [46, 30], security could still be achieved applying the WS-Security standards by signing SOAP-messages or attaching signed security tokens (with Kerberos) and encrypting XML-content. However implemented protocols do not necessarily provide any means of session management or transport security. Also they might not provide any predefined or common practices for authentication as HTTP does.

So it should be possible to implement secure ports in a way, that even those protocols can satisfy the security requirements and mitigate the described threats. So an authentication mechanism is designed, that clearly separates each step required on the server side, so that port implementation can override this custom common solution, with protocol specific standard processes.

Ports accept Service-request-objects with the containing the following data:

- method name
- arguments (as array)
- service properties
- meta-data (e.g. context)

This data structure is now extended by an additional field for the security token. The token contains an authentication object that can be compared to a ”claim“ in WS-security. It contains the username and some kind of credentials. It is intended to be passed on to one of the available authentication providers. If at least one of the providers can verify the validity of the credentials, authentication is successful.

The authentication process is not bound to a specific type of credentials. Examples for possible implementations are:

- Empty: No additional credentials, every user can claim an identity. This can be used for providing public access.
- Password: A plain text password or a hash thereof
- Public key with signature: similar to ”publickey“ authentication in SSH.
- Kerberos: A service authenticator combined with a ticket.

To ensure the confidentiality of the credentials (since it might be a plain text password), the whole message is encrypted. The encryption is implemented for use with any asymmetric algorithm. To detect replays of the same message, a timestamp is added to the encrypted content of the message (as recommended by WS-Security). Integrity of the message can be achieved by adding a checksum to the encrypted content of the message.

This authentication mechanisms however only works one way. The server has no means to send the client a response without risking confidentiality and integrity violations. For this purpose the client must include a response-encryption-method in the encrypted message content. The response-encryption attribute contains the name of the algorithm and required keys for the encryption. Some examples for response encryptions:

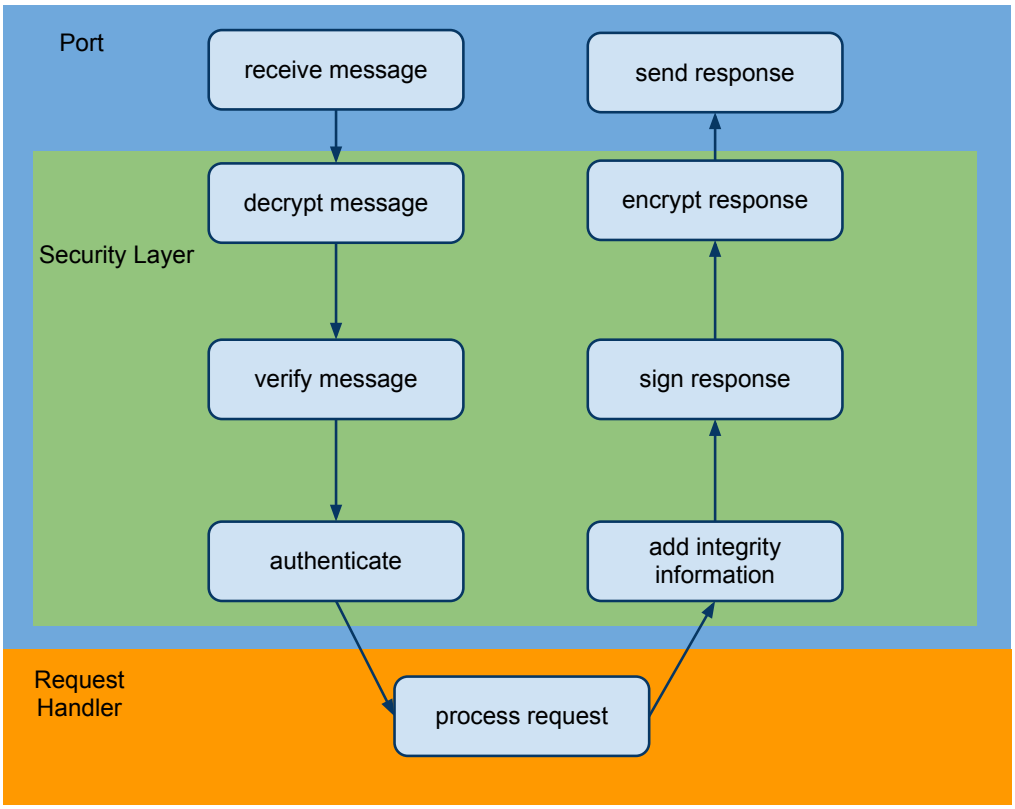

Figure 5.3: Position of the security layer in a Service Request

```
{ name: Caesar, key = ''3`` }
{ name: AES-256, key = ''A89683CB8...`` }
```

The resulting secure message now contains:

- original message
- username
- credentials
- timestamp
- checksum of the original message, username and the timestamp
- response encryption method

Figure 5.3 outlines the generic authentication mechanism for use with protocols that do not provide any security. The security layer involves several message processing steps. To properly modularize the message processing each step is encapsulated in a dedicated component. These

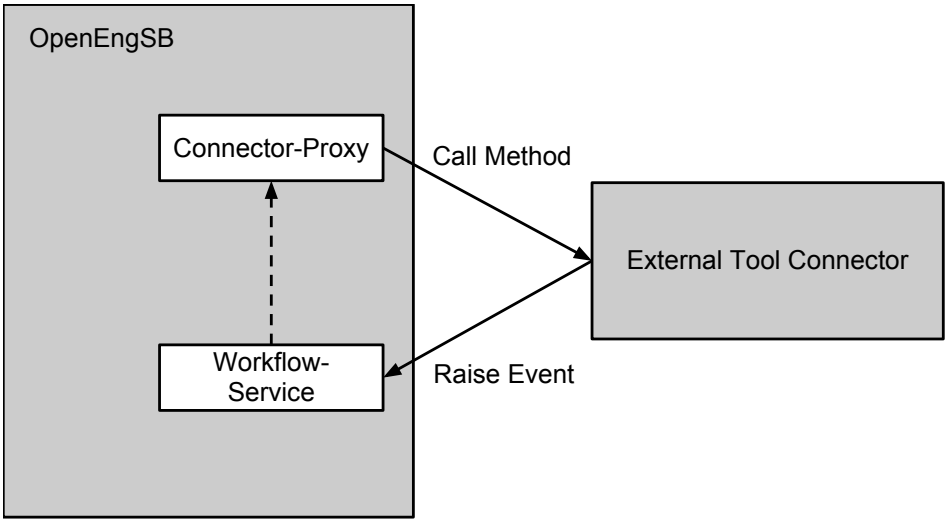

Figure 5.4: External connector interactions

components are then combined using the Chain of Responsibility Pattern [28]. One element in the chain is responsible for processing the message for the next element, but also to process the return value for the previous element. For example the same element is responsible for decrypting and encrypting a message. This makes it easy to change encryption algorithms as well as the whole encryption scheme.

External Tool Connectors

A running openengsb instance is considered a host for this connector. For the workflow engine and other systems interacting with this instance, it seems as if the tool connector is part of the same instance. The OpenEngSB transparently forwards invocations to some remote host using a specific port implementation that is configured for a specific connector proxy. Communication with external tool connectors is performed using the same port mechanisms as for the core services.

Figure 5.4 illustrates what kind of requests are exchanged between external connectors and the OpenEngSB. Method calls on the connector are transported to the external connector using similar principles as for sending them to the OpenEngSB. In the proxy connector a specific outgoing port instance is configured that is intended to be used for sending the request. The outgoing port is responsible for all aspects of the network communication when sending a method call request to the remote connector including security. In outgoing ports, similar to incoming ports, filterchains are used to modularize the processing of the message (see Figure 5.5).

External connectors must provide a way to receive messages from the host OpenEngSB. While the OpenEngSB is aware which user is invoking the proxy on its side of the communication, the security context the OpenEngSB uses to look it up is not available on the remote machine. So the OpenEngSB takes over the filtering of messages to the remote connector. External connectors are not required to perform their own authorization. Of course the incoming

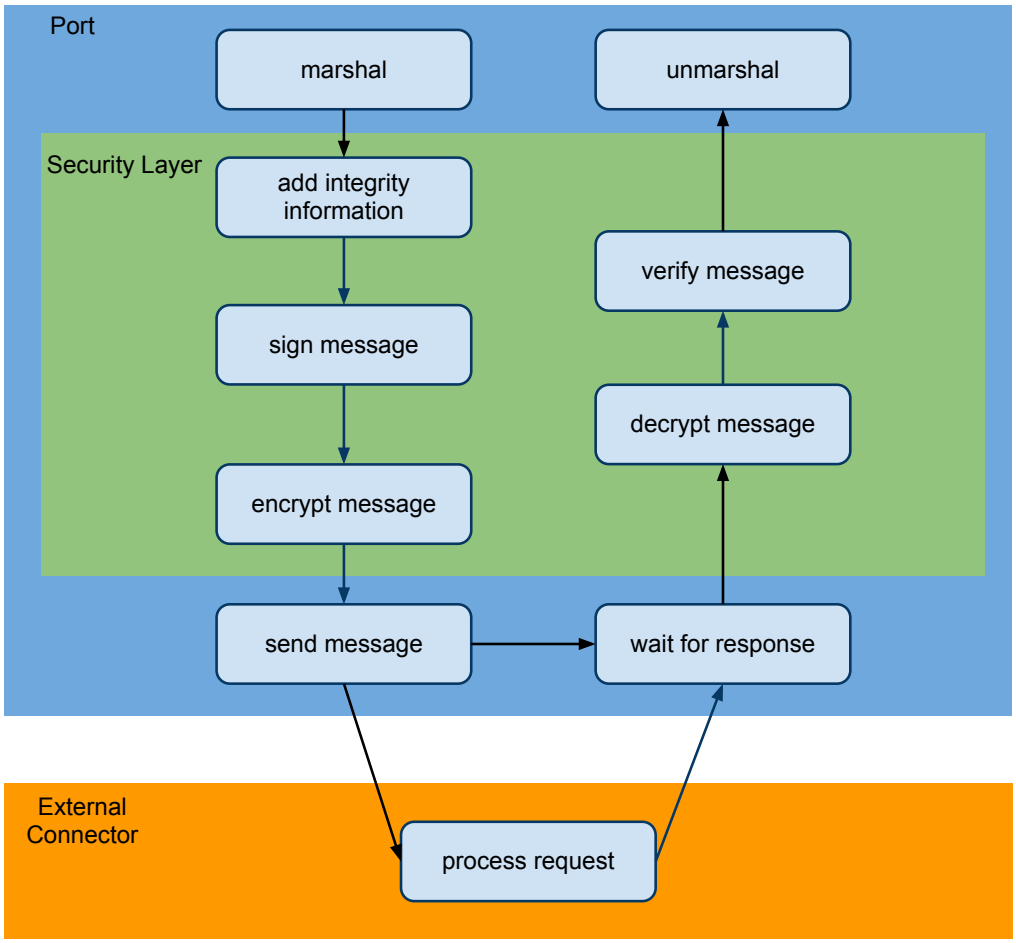

Figure 5.5: Outgoing port with security layer

requests must be verified to originate from the trusted OpenEngSB instance. Using the security mechanism described in Figure 5.5, the OpenEngSB only sends messages signed with its private key, so that any remote connector can verify the message using the public key.

Multiple OpenEngSBs

The same principles can be applied to a federated OpenEngSB environment. A simple solution for integrating two OpenEngSB instances is to exchange plain service requests and events (since they are handled with service requests). One OpenEngSB could also register one or more of its connectors as remote connector on the other OpenEngSB and thus providing an external connector. In all cases the techniques described previously can be reused.

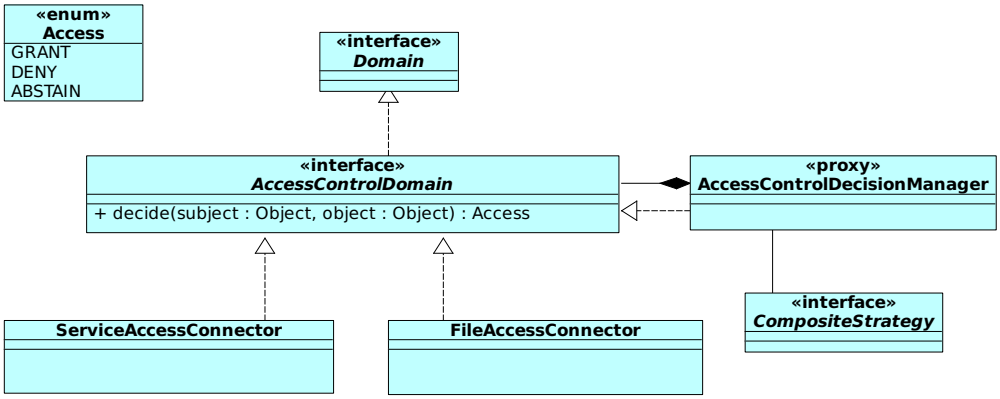

Figure 5.6: Access control domain architecture

5.4 Access Control

With authentication mechanisms in place everywhere it may be needed, the next step is to enforce authorization policies on the components. The issues Repudiation, Elevation of Privilege are usually addressed using access control mechanisms. The first challenge is to describe the policies and permission systems and will be addressed in chapter 5.4. Another is to actually enforce the described policies on the desired objects. Invocation on service methods are considered the only object class in this context. After that a solution for enforcing these policies is described.

Describing Policies

In section 2.9 the core principles and some domain specific languages for describing access control policies are described. In addition to the wide range of ways of describing policies, there may also be many different types of assets to protected (e.g. services, files, external resources). From all techniques for describing access control policies, the most generic approach is attribute-based access control. The specification of XACML shows that attributes can be used to express other access control concepts like hierarchical roles or security levels. So attributes are used to control access in the OpenEngSB too. Every resource that should be protected is assigned one or more security-attributes in some way (e.g. Java-classes with annotations). So access control decisions are based on the target assets' security attributes. The decision itself it then handled with a domain-connector mechanism similar as it is done for authentication (see section 5.2). An Authorization Domain is defined, that is able to make access control decisions about a subject on some object (see figure 5.6).

The object is the asset to protect. A connector implements some way to decide whether access should be granted on the object.

Particularly the domain defines two operations:

- supported: checks whether the connector is able to handle the supplied object.

- decide: makes the actual decision based on the connector implementation.

The result of a decision can be either of

- GRANT: the connector has a policy that permits access.
- DENY: the connector has a policy that denies access.
- ABSTAIN: the connector has no policy matching the given object.

An example for such an asset is a method invocation. To be able to decide access on a method invocation a connector that supports such decisions must be implemented. The is supplied with data about the methodcall (Classname, methodname, instance, arguments), and must decide some way whether the user should be granted permission to invoke the method. The connector can base its decision either on classname only, or the specific instance. In this example the supported-operation of the connector must check whether the supplied object is a method invocation.

There is no defined backend where the connector must store its policies. This way connector implementations may obtain the policies for their decisions from an arbitrary source. So it is possible to implement a connector that makes decisions based on XACML-description files, or in some custom relational database. In chapter 6 a set of default connectors will be introduced along with a default backend for storing permissions.

Integration environments will require multiple access control connector instances at the same time to make all necessary decisions. So like for authentication, a strategy for handling multiple authorization connectors is required.

A simple strategy would be to grant access as soon as one connector decides the user is permitted to. Example: There is one access control connector that controls access to the services and one that control access to files. Every time a decision is requested, either of the connectors must yield a positive result to permit access.

An alternative strategy could require all or a majority of the connectors that support the object to yield a positive result.

A set of default strategies is described in chapter 6.

Enforcing Access Control

With a source for access control decisions and policy descriptions, the next step is to enforce the policies. For every entity that is to be protected a decision point has to be defined and enforced. The most important entities touched in this thesis are services and user interface components. So every request for a service invocation or an user interface action must be intercepted, checked for proper authentication and authorization. This goal of the interceptor is to mitigate the threat of elevation of privilege.

To enforce the policies in the user interface, framework specific mechanisms must be used. To ensure service requests are intercepted, the ”proxy pattern“ is used (more specifically ”Protection proxies“) as defined by GOF [28] (see Figure 5.7).

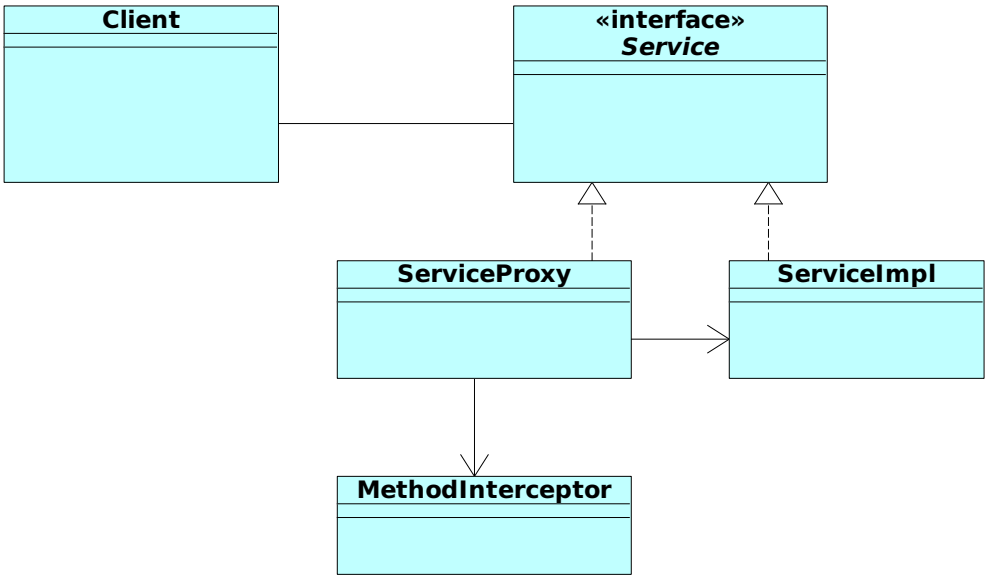

Figure 5.7: Service with method interceptor

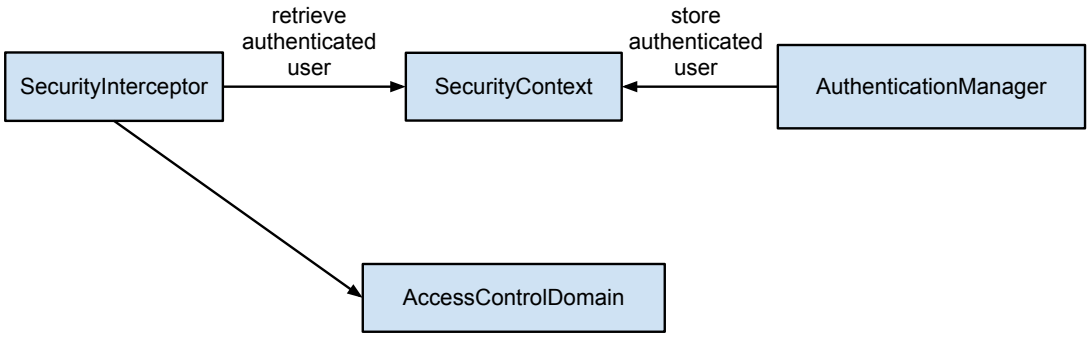

Figure 5.8: Usage of the Security Context

Every service object, that is to be protected by access control policies is not registered as a service itself. A proxy must be created that invokes a ”Security-interceptor“ prior to each invocation.

The security interceptor retrieves the authenticated user from the Security context and delegates the access decision to the corresponding internal domain service (see Figure 5.8). If access is denied, the interceptor throws an exception. If the invocation originated from a remote service request, the exception is sent as the result of the service call.

When services are to be protected by access control policies, only the secured proxy object must be registered as a service in the OSGi-registry. So when the service registry is queried for a service, only the secured version can be discovered (see Figure 5.9).

Also all service invocations are passed to the security interceptor first. This seems straight

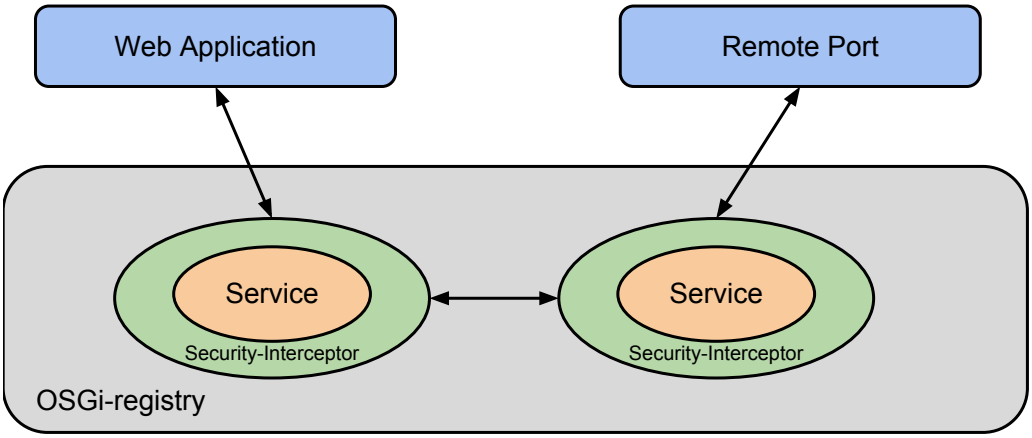

Figure 5.9: Usage of the Security Context

forward for remote invocations received by e.g. message oriented middleware. However the same applies for internal service calls, that may be the consequence of an event raised by some other service. Also method calls in the context of internal administration tasks must go through the security interceptor.

Domain specific access control

The domain and connector concept works great for generic concepts like method invocations. However taking the methods parameters into account is a difficult, if not impossible task for a generic connector that handles service access. The parameters are often used to identify a specific artifact. There is no way for the generic interceptor to know the semantics of each operation and its parameters.

The most suitable place to define these semantics is the definition of the domain. For this sematics to have a meaning the Security interceptor has to take them into account too.

So a domain may define custom access control decision points that take the parameters into account. The security interceptor must check whether such custom services are associated with the intercepted method call and enforce the domain specific policies.

5.5 Tools with complex security

There are many tools that ship with a tool specific security layer and thus require additional authentication. For example the Git SCM uses the user- and permission management of the underlying operating system to decide about read and write access to a repository. So if a Git repository is set up without any means of anonymous public access, the user must authenticate for any action on the repository. In a (Software+) Engineering environment this tool specific security layer and thus the tool specific authentication process should be hidden to the user. This

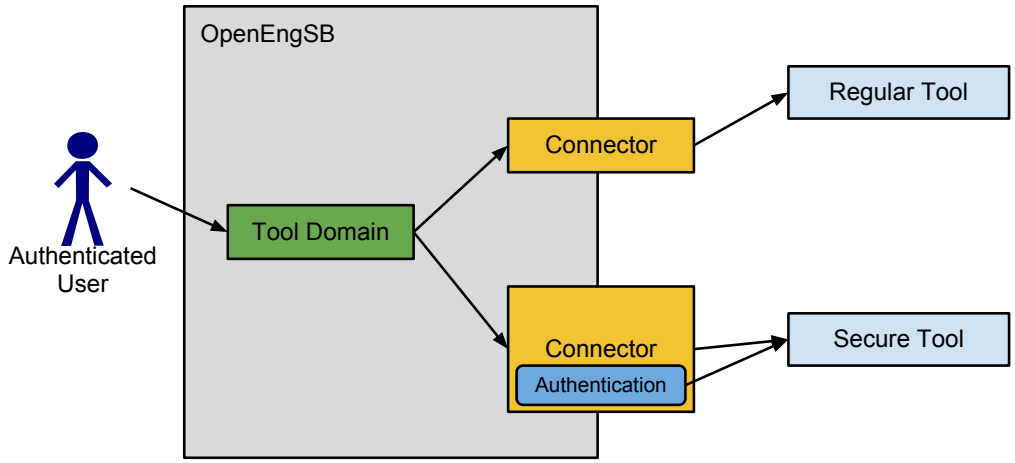

Figure 5.10: A regular and a secure tool that requires authentication as they are conncted with the OpenEngSB

means, when the user intends to retrieve a file from the SCM, the specific tool used for SCM should be transparent (see Figure 5.10).

This means the tool connector is responsible for implementing the tool's authentication mechanism. In order to authenticate with the tool, credentials are required. One possibility would be to reuse the credentials supplied by the user when he authenticated with the OpenEngSB. This approach is very limited as the credentials required for authentication with the OpenEngSB and with the tool may not be compatible (e.g. password and public key or fingerprint). Even if both tools rely on passwords, they might use different hash algorithms making it impossible to forward the credentials.

Another approach is, to provide the connector instance with a static set of credentials, that is used in any invocation. This means that the connector configuration contains attributes to hold the credentials. So from the tool's point of view all actions are performed by the same user. This approach has a low impact on connector design, so it is easy to implement. It is also easy to deploy. The access control is performed by the OpenEngSB's authorization layer.

This solution is suitable for internal integrations where the OpenEngSB can be trusted with the decision about access to the connector. However, if the connector is running at an external company, it might not trust the company running the OpenEngSB. Also programming mistakes in the connector implementation may grant access to a wide range of data and functionality in the tool.

A more secure approach is to associate users with their own credentials. However, the user cannot be forced to authenticate with the OpenEngSB with these credentials. The user must maintain a set of associated credentials for tool specific authentication mechanisms. This way, the OpenEngSB only serves as a bridge to the external connector, and the control about authorized users remains on the other companies side. In addition, when the credentials are associated with users, it is possible to store them in a secure container that is protected with the users orig-

inal OpenEngSB credentials. This is achieved by providing a credential management service based on the concepts introduced in section 2.7.

CHAPTER 6

Prototype Implementation in OpenEngSB Framework

6.1 Existing Frameworks

Before implementing anything several framework have been analyzed. This analyzes takes into account how the authentication and authorization are handled in general, and how the frameworks could be used to implement the concepts described in chapter 5.

Java Authentication and Authorization Service (JAAS)

JAAS [38] was introduced as an optional package (extension) to the Java 2 SDK in version 1.3. It can be used for authentication to determine who is running code and for authorization to ensure the user has the permission to run the code. The authentication component of JAAS can be extended by custom authentication plugins without modifying the application itself. There are plugins for providing simple password authentication as well as Kerberos integration using a GSSAPI [40] plugin. The authorization decision point is the SecurityManager that is deeply integrated in the class loading process of the Java Virtual Machine.

The authentication part of JAAS supports a pluggable architecture making it easy to implement additional authentication-modules. In order to implement a custom authentication module, the LoginModule interface must be implemented. The specification of the Login-Module interface also considers two-phase-authentication mechanisms by separating login, and commit. While login just checks the credentials for validity, the commit operation adds the associated principals and permissions to the returned subject.

The permissions associated with the authenticated principals are then associated with an Access-Control-Context. This context can be used to execute privilaged actions using the AccessController class.

Configuration for JAAS is done in files, that are passed as system properties when starting the Java runtime environment.

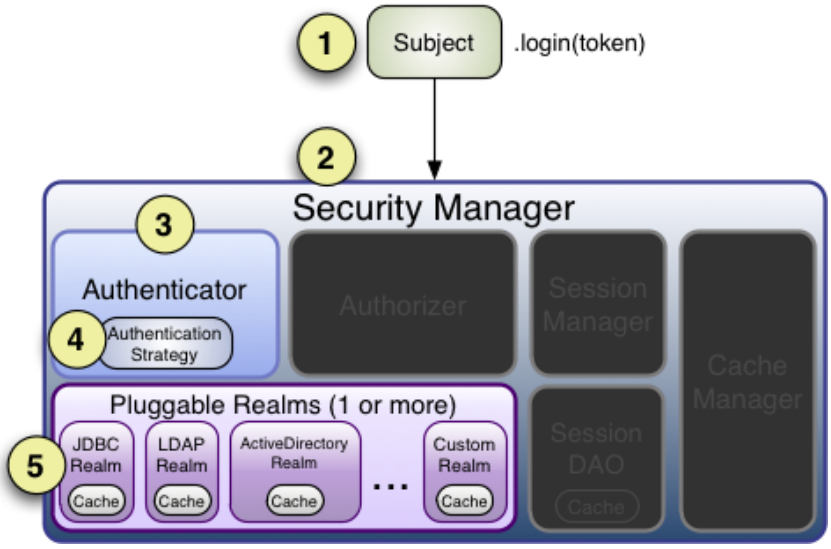

Figure 6.1: Shiro Authentication Sequence.[2]

While the capabilities and flexibility of the authentication in JAAS is similar to those described in the AuthenticationDomain, the authorization in JAAS focuses on executing code, and does not take other context information into account (e.g. which instance of the class it runs on). Also it requires the authentication-modules to provide the necessary permissions, which makes it impossible to define policies separately as required.

Apache Shiro

Apache Shiro [1] is a Java security framework that provides functionality for managing authentication, authorization, cryptography, and session management. Both authentication and authorization can be configured using pluggable data sources.

The authentication handles through a central Authenticator instance. Different authentication mechanisms are supported by separating them into realms (see Figure 6.1).

As opposed to other approaches, the architecture is built around the subject instead of the managing classes. Instead of supplying a managing class with the subject, the subject resolves the authenticator and handles the login. Similar to JAAS a realms provide a way of plugging in new authentication methods. The authenticator is configured with a authentication strategy that controls which realms to use (see Figure 6.2). This way implementing strategies similar to the ones described in section 5.2 is possible.

Shiro takes a similar approach to authorization. It uses the same realm architecture for providing multiple authorization methods. Custom realm implementations also allow implementing custom permissions, which would allow defining permissions on specific service instances or

[1] https://shiro.apache.org/

[2] https://shiro.apache.org/authentication.html#Authentication-sequence

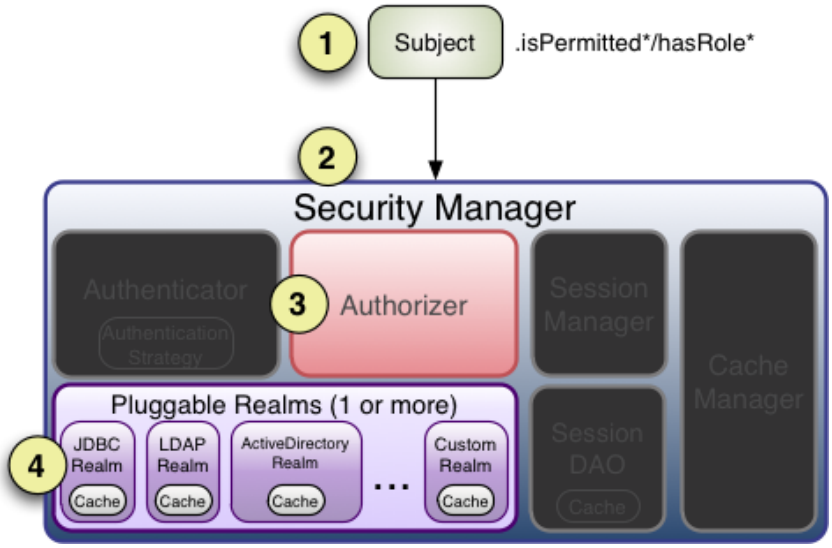

Figure 6.2: Shiro Authorization Sequence.[3]

operations.

Shiro also provides several interception techniques to enforce access control:

- Annotations: Methods can be enhanced with annotations that contain security assertions (e.g. a method requires the permission "create account"). To enforce the annotations the application must have Aspect Oriented Programming (AOP) support. Shiro allows Spring AOP[4], AspectJ[5] or Guice[6] to be used for this task.
- JSP tags: A tag library that provides tags for conditional rendering of page components when using Java Server Pages.
- Manual: An application may also check for the permissions directly and act accordingly using if-statements.

The configuration can be performed in a file (see Figure 6.3, that is loaded at startup and associated with a global Security Manager. The Security Manager is a singleton class. The instance is constructed when loading the configuration. In a web application the path of the configuration file is passed with a filter in the file "web.xml" which is used by Java EE applications to configure the container.

One problem with the realms in shiro is, that realms for Authorization and Authentication are tightly coupled. A realm capable of authorizing an action must be able to authenticate a user as well (due to the interface hierarchy in the framework).

[3]https://shiro.apache.org/authorization.html#Authorization-AuthorizationSequence
[4]https://www.springsource.com/
[5]https://www.eclipse.org/aspectj/
[6]https://code.google.com/p/google-guice/
[7]https://shiro.apache.org/configuration.html#Configuration- SecurityManagerfromanINIinstance

```
[main]
sha256Matcher = org.apache.shiro.authc.credential.
   Sha256CredentialsMatcher

myRealm = com.company.security.shiro.DatabaseRealm
myRealm.connectionTimeout = 30000
myRealm.username = jsmith
myRealm.password = secret
myRealm.credentialsMatcher = $sha256Matcher

securityManager.sessionManager.globalSessionTimeout = 1800000
```

Figure 6.3: Example for Shiro INI configuration file. [7]

Spring Security

Spring Security is an authentication and access-control framework that is designed to work closely with the Spring framework[8]. It claims to be the defacto standard for securing spring-based applications.

To perform authentication in a spring security based application, an authentication manager is used. An authentication manager can be configured with one or more authentication providers. The providers each represent an implementation of a specific authentication mechanism (see Figure 6.4). The authentication manager attempts authentication with each associated provider until a successful authentication is returned. The behavior of the authentication manager can be customized by implementing the corresponding interface.

Authorization is handled by an "Access Decision Manager". It is associated with a set of "voters". A voter is an implementation of some policy to decide about access. The access decision manager decides how the result of the voters should be aggregated into a final decision. Spring Security provides three strategies by default:

- Affirmative based: As soon as one voter votes to grant access, access is granted
- Consensus Based: The majority of the associated voters must vote to grant access.
- Unanimous based: All associated voters must vote to grant access.

To enforce access control, provides interceptors that can be associated with objects using AOP. This can be done using Spring's own AOP framework, or with AspectJ[9]. Annotations on the methods serve as a base for authentication decision for the voters that decide method invocations.

To make the interceptors aware of the authenticated user, Spring Security defines the security context. It is a thread local storage that contains the current authentication token. Authorization decisions are made for the user stored in this context.

[8] http://www.springsource.org/
[9] http://www.eclipse.org/aspectj/

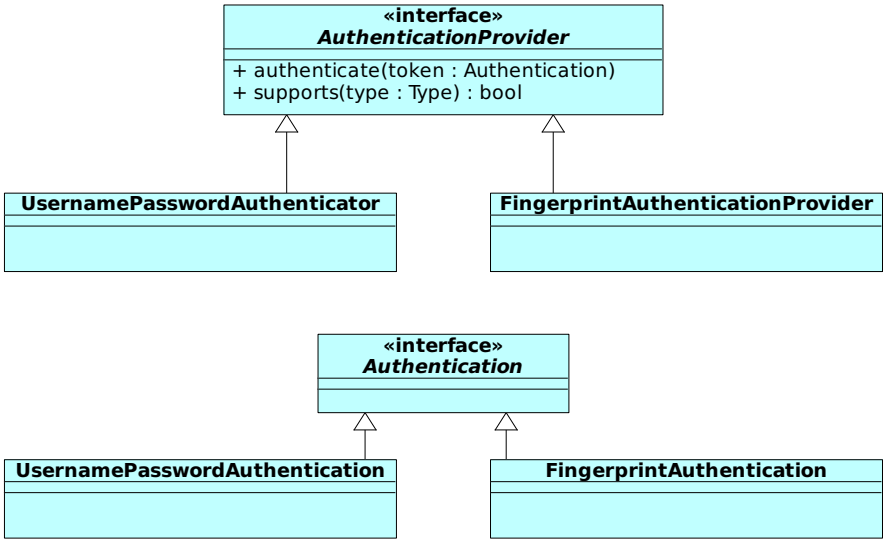

Figure 6.4: Spring Authentication Manager model

The configuration of the security relevant elements described above takes place in files and are injected to Java objects using the provided dependency injection framework (Spring Beans). The dependency injection process is handled either by the OSGi container itself, or the corresponding web application framework.

Conclusion

While all frameworks support changing the configuration of authentication modules at runtime, JAAS lacks the flexibility to change the source for authorization policies. Since authorization decisions in the OpenEngSB require to take context and the specific object instance into account, JAAS is not a suitable framework to implement security. Shiro and Spring Security support dynamically changing authentication and authorization components. The use of AOP to intercept method calls and the possibility to use custom interceptors makes it possible to delegate the authentication and access control decisions to OSGi services, which can be changed in a very dynamic way.

Spring Security provides a very well documented and highly customizable API. However the configuration is tightly coupled to Spring's dependency injection framework. It is very well integrated in Java EE web applications, but not in OSGi environments. The OSGi compendium defines the blueprint container [57] for dependency injection. The Spring framework relies on its own implementation of the blueprint container Spring Dynamic Modules[10]. However the implementation is not finished and no longer developed. Shiro implements its own lightweight

[10] http://www.springsource.org/osgi

```
public interface AuthenticationDomain extends Domain {
    Authentication authenticate(String username, Credentials
        credentials) throws AuthenticationException;
    boolean supports(Credentials credentials);
}
```

Figure 6.5: Authentication domain definition

configuration mechanisms, that can easily be configured in other blueprint container implementations (e.g. Apache Aries[11]. However the inability to separate authentication from authorization modules decreases the flexibility of the framework significantly.

6.2 Authentication

Implementing the concept from section 5.2, authentication is handled by connectors in the authentication domain. The authentication domain is defined as a Java interface containing two methods “authenticate” and “supports”.

The Credentials interface is a flag interface that indicates that the supplied object is designed to represent credentials. Authentication connectors provide their own credential models by registering the class object to a registry. This registry is used to convert the credential objects from any serialized format back to a Java object.

Entry points for authentication are remote procedure calls and the web application. In both authentication is handled differently. Every entry point holds a OSGi service query that is used to discover the authentication connector to use. While it is possible to let this filter point to a single connector, it should point to a virtual connector representing the composition of multiple authentication connectors.

Consider the following example where the entry point is associated with the following filter:

```
(location=authentication-root)
```

This means the service where the value of the property “location” is either the exact string “authentication-root” or a collection containing exactly that string is resolved and invoked every time an authentication request is received (see Figure 6.6).

After a successful authentication the authentication object returned by the responsible connector is stored in a threadlocal store. The implementation reuses the security context provided by the Spring Security framework.

Entry points

The two main entry points where authentication is done are the ports receiving remote service requests and the web application. Web applications usually provide some web page that prompts for username and some credentials (most commonly a password). The web application then

[11] http://aries.apache.org/

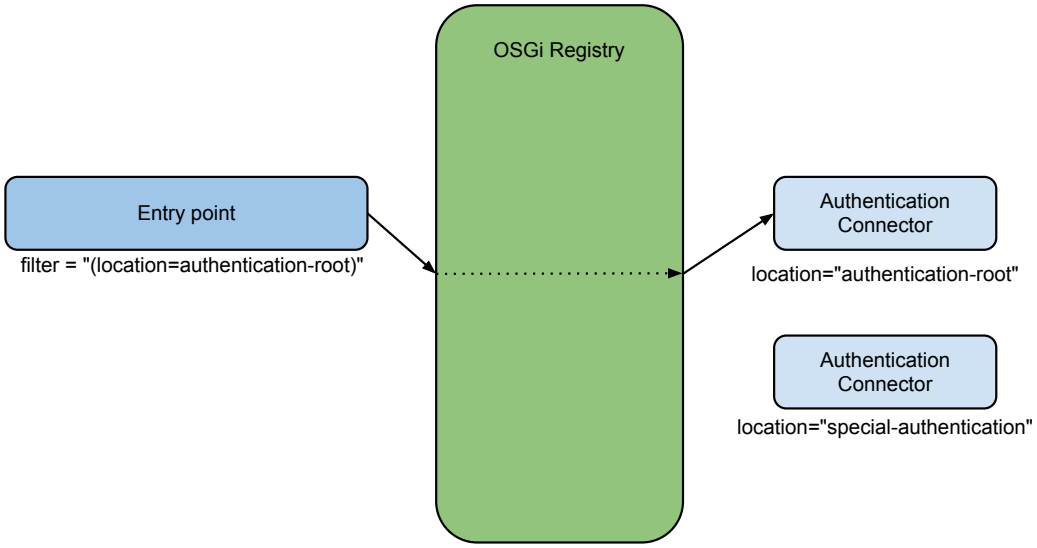

Figure 6.6: Authentication connector discovery

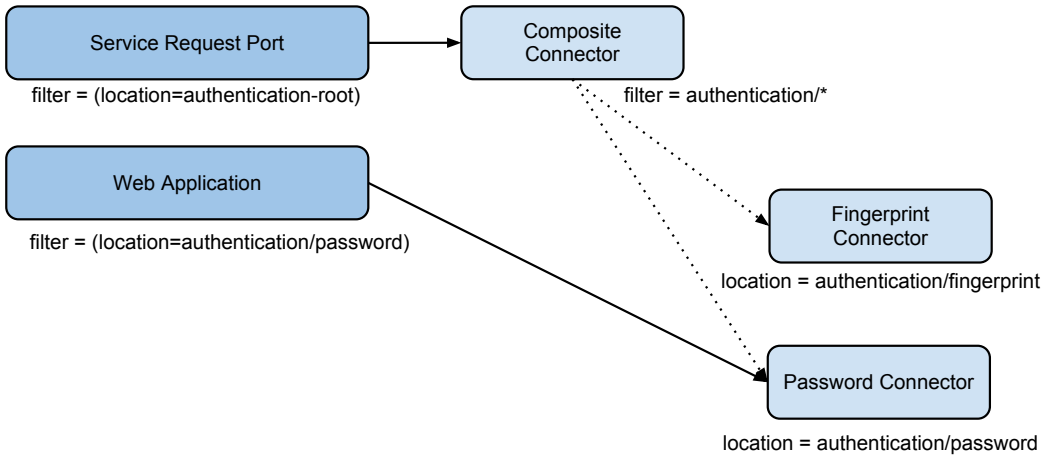

Figure 6.7: Authentication connectors at different entry points

must provide some kind of bridge between the web page and the authentication connector. It is possible that the web application does not support entering all types of credentials supported by the authentication connectors used. So a reasonable approach is to use a different connector location for that (see Figure 6.7).

This reduces the overhead of calling connectors, the web application cannot support anyway.

In the administration user interface provided with the OpenEngSB, the default behavior of the authentication page uses this mechanism. The bridge between this web application and the

```
public interface AuthorizationDomain extends Domain {
    Access authorize(String user, Object object);
    boolean supports(Object object);
}
```

Figure 6.8: Authorization domain definition

authentication connector, is realized using Shiro components. The filter configuration of the application points to a shiro configuration file. This causes a filter to intercept all requests to the web application and check for proper authentication the realm defined in the configuration file. The realm is a custom implementation of the "AuthenticatingRealm" interface where the authenticate-operation queries the responsible connector from the OSGi-registry and forwards the credentials. This does not take advantage of the pluggable realm architecture of Shiro, since the plugin mechanism is handled by OpenEngSB domains.

In service request ports the messages received must contain authentication information the port than can use to pass on to the corresponding authentication connector. The authentication mechanism and protocol for remote service requests is described in more detail in section 6.5.

6.3 Authorization

Authorization decisions are handled by connectors implementing the Authorization Domain. The domain interface definition defines the operations "authorize" and "supports". The result of the "authorize" operation can be one of the three "GRANTED", "DENIED" or "ABSTAINED". Access to a resource is only granted when the result of the decision is "GRANTED". When using a single authorization connector instance to make all decisions other results cause the access to be denied. However in tool integration environments more than one connector is required. The discovery of the required Authorization connector follows the same mechanics as the discovery of Authentication connectors at authentication entry points.

It would be possible for an authentication point to use the authentication object and perform access control on its own before invocation. This has several disadvantaged. The approach is very error-prone because the check can easily be forgotten. Also this way nested calls cannot be handled. So if the user is granted permission to invoke a service that later invokes a service that the user is not authorized to access, there is no controlling instance preventing this.

So it is important for security to be handled transparently using interception points. The most important interception point is at the service level. As described in section 5.4 services are protected by hiding them behind a proxy that makes sure the Security Interceptor is called before each invocation. The implementation of the interceptor does not methods declared by the "java.lang.Object" class as this causes several unexpected Exceptions and issues in third party frameworks. For example logging frame works in at the registry level cause the query process to fail in case calling "toString" throws an Exception.

The security interceptor is implemented using the MethodInterceptor interface as defined by

```
public interface MethodInterceptor extends Interceptor {
    Object invoke(MethodInvocation invocation) throws Throwable;
}
```

Figure 6.9: MethodInterceptor as defined by AOP alliance

the AOP alliance [12].

The "MethodInvocation" interface defines methods for providing the necessary metadata about an invocation:

- Target object
- Method name
- Arguments

This interface is used as the model for authorization connectors. So every connector that is intended to make decisions about access on a service request must support this type of object because the "supports" and the "authorize" operations are invoked with an instance of MethodInvocation as object. The connector can use the object to derive additional metadata for the decision. If the associated authorization connector does not return a "GRANTED" result, the security interceptor raises an "AccessDeniedException" instead of allowing the invocation to proceed.

The other important interception point is in the user interface itself. The goal is, to cause user interface components, the user is not supposed to interact with for security reasons, to be hidden. The purpose of this is not meant to be additional security, but an improvement in usability. The backing business logic of the application is supposed to be handled by OSGi-services and thus uses the Security Interceptor for proper authorization.

In Wicket every component in the user interface is represented by a Java Object. The built-in authorization mechanism defines two actions "render" and "enable". A newly defined UIAction class serves as the common model for the authorization domain (as MethodInvocation is for service requests). It provides the connectors with the component and the action that is about to be performed on the component.

To make the authorization decisions easier for the connectors all classes, methods and instances may be associated with additional metadata in the form of attributes. Access control decisions can then be based on the authenticated user's and the given object's security attributes. For this purpose the SecurityAttribute annotation is defined in the OpenEngSB's API. It allows associating elements with key-value pairs representing attributes. This is also a good way of assigning human readable names to the protected objects (e.g. as in Figure 6.10).

Using the annotation only enables assigning security attributes at compile time. In order to associated specific instances with additional security attributes, a storage service for these attributes is introduced. An important restriction is, that such a service must only contain information about instances, managed by the same OSGi bundle, since bundles can be restarted

[12] http://aopalliance.sourceforge.net/

```
@SecurityAttribute(key = "org.openengsb.ui.component", value = "
    USER_ADMIN")
public class UserEditPage extends BasePage {
    ...
```

Figure 6.10: Example for a security attribute

```
public interface SecurityAttributeProvider {
    Collection<SecurityAttribute> getAttribute(Object object);
}
```

Figure 6.11: Interface for the Security Attribute Provider

separately. If a service would contain attributes about objects managed by another bundle, and the service's parent bundle is restarted, the stored attributes are lost. For this reason, the service interface only provides read access to the attributes (see Figure 6.11)

Elevated Privilages

With every service proxied with an interceptor this means that these services may also receive internal calls, initiated by a core component. An example for this is the connector deployer service. It watches a directory, and as soon as a connector configuration file is stored, it is parsed and deployed as a connector instance.

The registration of connectors is handled by the connector manager service. It is secured with the security interceptor thus requiring an authenticated user with the corresponding permissions to execute it. The connector deployer is a background task meaning it is not run by any user. For this purpose a special authentication token is introduced: The SystemAuthentication. If the security interceptor detects this special token in the security context, the invocation always proceeds.

Since this circumvents all security the interceptor provides, the threat of elevation of privilege becomes imminent. There must be no way for users to store such a token manually as authentication. For this reason this token is not created by a connector, but only directly. So if a component needs to perform a task with elevated privileges, this can be done by creating a SystemAuthentication token and storing it in the security context. After performing the action the token must be removed from the security context again (or replaced with the original authentication). A component cannot do this directly since this bears a security risk. The action requiring the privileges must be contained in a separate Runnable or Callable object. This object is passed to a function that makes sure the task is executed with the desired permissions. This is done by wrapping the action in a proxy that stores the SystemAuthentication token in the security context, executes the defined action and removes the token from the security context again. This proxy is executed in a separate thread, so that it does not influence the security context the calling component.

```
SecurityContext.setAuthentication(new SystemAuthenticationToken());
try {
    return originalAction.call();
} finally {
    SecurityContext.clear();
}
```

Figure 6.12: Execute an action with elevated privileges

```
    @DomainAuthorization("scm-repository-file")
    byte[] get(String file);
```

Figure 6.13: Method with domain-specific security handler declared.

Domain specific Access Control

As described in section 5.4, the definition of domain specific access control must be provided by the domain itself. So the domains require a defined way for implementing their access control. For this purpose the security interceptor is extended. After successful authorization, the method is checked for associated domain specific access control handlers. If one is found, it must return a "GRANTED" result too, for the invocation to be successful. In order for the domain to define the specific access control handler to use on a method the "DomainAuthorization" annotation is introduced. It can be used on either specific methods or on the entire interface definition. In the annotations value, the name of the service that handles the decision is stored (see Figure 6.13).

The referenced service implements an interface similar to the AuthenticationDomain, but without the supports-operation. Since it was declared on a method explicitly, it can be assumed that it is supported. The security interceptor checks a method for the annotation to be present. The handling service then can read additional security attributes from the method to make a decision.

6.4 Administration

This modular approach to authentication and authorization can make administration of permissions and policies challenging and uncomfortable for system administrators and project managers. To provide a central authority for authentication and authorization connectors for managing the relevant user specific data, the UserManager service is introduced. The interface defines methods for creating and deleting users. Further methods are defined for associating a user with credentials and permissions.

A user can be associated with several credential values. For example a user can have one of type password with a hashed password as value, and another of type "fingerprint" with a reference image as a value. The connectors obtain a reference to the user manager to lookup users in authentication requests and compare the supplied credentials with the stored ones. This is

Figure 6.14: Users and credentials

Figure 6.15: Users and permissions

however not mandatory, as connectors may retrieve additional reference information for authentication another way too. For example it can delegate the authentication to an external entity like an directory server.

A similar approach is taken for administrating permissions. Along the lines of XACML, the definition of permission is done in permission objects that can be combined in permission sets. Permission sets may contain several permissions and other permission sets (see Figure 6.15).

This way hierarchical RBAC can be realized by modeling roles as permission sets (similar to the RBAC profile of XACML). Permissions are defined by a set of attributes represented as key-value pairs. The values may either be single values (numbers, strings) or lists of such values.

Authorization connectors define their custom permissions as plain Java objects. These objects are mapped to a list of key-value pairs by the user manager service. An implementation of the service is given using the Java Persistence API (JPA) to manage the user data in a relational database.

For the service authorization connector, a permission definition contains the following attributes:

- Type: The type of service (identified by the interfaces name or the corresponding SecurityAttribute)
- Instance ID: The persistent identifier of a service instance.
- Operation: The name of the operation performed on the service. The name can be either the method name in the interface definition or defined in security attributes on the method.
- Context: The context the operation is invoked in.

All values are stored as strings and are optional. This way permissions of variable granularity can be managed.

The connector responsible for user interface actions uses these attributes:

```
public interface PermissionProvider {
    Collection<Class<? extends Permission>>
        getSupportedPermissionModels();
}
```

Figure 6.16: Permission Provider interface

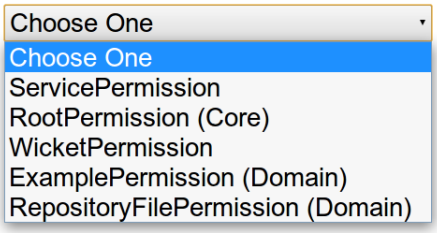

Figure 6.17: List of available permission models in a User Management UI

- Component name: The name of the component that the action is to performed on. In wicket the classname of the component can be used for this purpose. The generic approach is to register the name of the component with a security attribute provider.
- Action: The action to be performed. The Action can be one of “RENDER” and “ENABLE”.

To ease the administration of the variety of available permissions, all connectors provide their custom permission models to a central registry. An administration interface can lookup all permission types and their corresponding model to provide an interface for assigning new permissions to users and permission sets. Domains defining custom access control mechanisms also use custom attribute-based permission models. These models are also registered with the same registry.

Like for the security attribute providers, the OSGi-registry itself is used to realize this registry. In order to provide a permission model to the registry, a service implementing the PermissionProvider interface is registered.

Bundles providing permission management functionality query the OSGi-registry for all services implementing this interface to get a list of available permission models (see Figure 6.17).

6.5 Remote Infrastructure

With an established generic model for authentication and a transparent way to enforce access control on service request, the next step involves transmitting the requests to the OpenEngSB. In web applications the best way to make the communication secure is by encrypting the channel with SSL/TLS [17] using the HTTPS protocol [51]. This way credentials and other sensitive

```
{
   "methodName":"executeWorkflow",
   "classes":[
      "java.lang.String"
   ],
   "args":[
      "simpleFlow"
   ],
   "metaData":{
      "serviceId":"workflowService",
      "contextId":"foo"
   }
}
```

Figure 6.18: Sample service request

information is transmitted securely. The host's identity can also be verified by checking the certificate.

For remote service requests exchanged through ports in the OpenEngSB, this must be handled differently. The ports can implement potentially any protocol even if it does not provide any support for security. The default implementation in the OpenEngSB uses JMS as a transport protocol. Not all JMS vendors provide security [55], so the most viable option is to implement custom message-level security. In section 5.3 the process of securing the messages is outlined.

Incoming Service Requests

When remote applications intend to send service requests to a host OpenEngSB, the first step is to generate a session key. It is generated for use with a previously defined symmetric encryption algorithm. The session key is used to encrypt the request. The key itself is encrypted using the host OpenEngSB's public key. An example for a method request encoded in JSON is shown in Figure 6.18.

The client application must also authenticate with the OpenEngSB. So authentication information is added to the request (see Figure 6.19).

The resulting message is then encrypted with the generated session key. The session key is then encrypted with the public key of the host OpenEngSB. The resulting encrypted message (Figure 6.20), is then wrapped into a JMS Text message and sent to the receiving queue of the OpenEngSB.

On the host side the message goes through the cycle described in Figure 5.3. The JMS port in the OpenEngSB first parses the encrypted message by decoding the JSON-message and separating the encrypted key from the encrypted payload. The session key is then decrypted using the servers private key, to later decrypt the message. As the decrypted message is in JSON format again, the message needs to be parsed again.

Before authentication takes place, the message is verified using the supplied checksum and timestamp. The timestamp is used to detect replayed messages. A message sent by a user must

```
{
   "principal":"admin",
   "credentials":{
      "type":"password",
      "data":{
         "value":"admin-password"
      }
   },
   "timestamp":1317995751,
   "message":{ ... }
}
```

Figure 6.19: Service request with security data attached

```
{
  "message":"tcrvJ/6Kgz[...]UVt7Q8Vvol",
  "key":"kbM3XdBhzd[...]oCD3UPzA=="
}
```

Figure 6.20: Encrypted secure request

```
{
  "type" : "Object",
  "className" : "java.lang.Long",
  "arg" : 1
}
```

Figure 6.21: Wrapped result of an operation

always contain a timestamp that is higher than the last message sent by the same user. The timestamp cannot be modified by a man in the middle because it is encrypted when sent across the channel. Optionally, the port can also reject messages when the timestamp is significantly lower than the current time, i.e. if the message is too old.

If the message is accepted as valid, the authentication information is extracted. The port then obtains a reference to the responsible authentication connector and tries to authenticate the message. In case of a successful authentication, the resulting authentication token is stored in the security context.

The actual invocation is then handled by the Request handler service. It reads the parameters for discovering the service and the context to execute the operation in from the supplied meta-data. The result of the operation or in case of an exception the error message is then wrapped into a result object (see Figure 6.21).

After adding integrity information (timestamp) to the resulting message is encrypted using the session key provided with the initial request. The encrypted result is then sent back to the

```
public interface ConnectorRegistry {
    void registerConnector(String instanceId, byte[] key, String
        algorithm);
}
```

Figure 6.22: Connector Registration Service interface definition

client.

After receiving the response message, the client decrypts it with the session key generated for the request. Like previously on the server side the timestamp is verified to protect against replay attacks.

The outgoing scenario is similar with one key difference: As the client does not maintain its own public/private keypair, the host cannot use it to encrypt the generated session key. This means the remote client must register with the host OpenEngSB to be able to receive messages. This enrollment process involves generating a symmetric encryption key on the client side. The key is encrypted using the host's public key, and wrapped in a service request targeting the connector registration service in the OpenEngSB. The definition of the connector registration service's interface is shown in Figure 6.22.

The connector registration service is a regular service provided by a core component, and is protected by the security broker. This means the registration request must include proper authentication information and the authenticated user must be authorized to register a new remote client. Because no unauthorized user can register a new connector, a possible flooding attack cluttering up the registry with dummy registrations is not possible.

When a remote client is registered, the OpenEngSB can securely send messages by encrypting them with the registered key as the session-key. The plain version of the service requests transmitted to the client follow the same format (see Figure 6.18). The encrypted version does not need to be wrapped in another message, since no encrypted session-key is transmitted along with it.

Modularization of the Gateway

The process described above is intended to be a secure default behavior. As stated in RQ1 (section 3.1) each part of the process should be exchangeable and open to company specific adoptions. So each step in the process must be encapsulated in a separate component that can be to make it exchangeable and reusable. As designed in section 5.3 this is achieved by using the Chain of Responsibility Pattern [28]. For this purpose a FilterAction interface is defined:

Every filter is responsible for both transforming the request, and transforming the result. This means the Filter that unmarshals the request from a JSON object to a Java Object, is also responsible for marshalling the resulting Java Object to JSON format again. When combining filters to form the processing pipeline in a port, the configuration results in a pyramid like structure (see Figure 6.24).

This way, each processing step can be configured separately which allows easier exchange of certain aspects, like the message format. In order to exchange the message format to for example

```
public interface FilterAction {
    Object filter(Object input, Map<String, Object> metaData) throws
       FilterException;
}
```

Figure 6.23: FilterAction interface

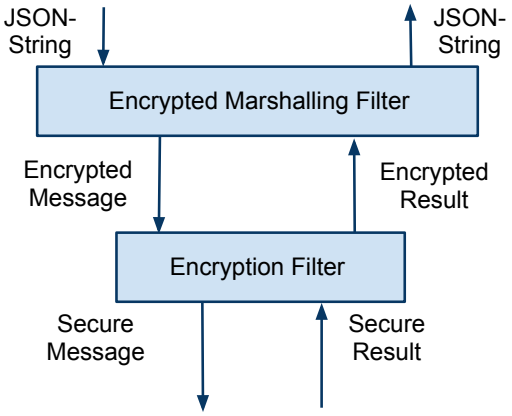

Figure 6.24: Filter pyramid

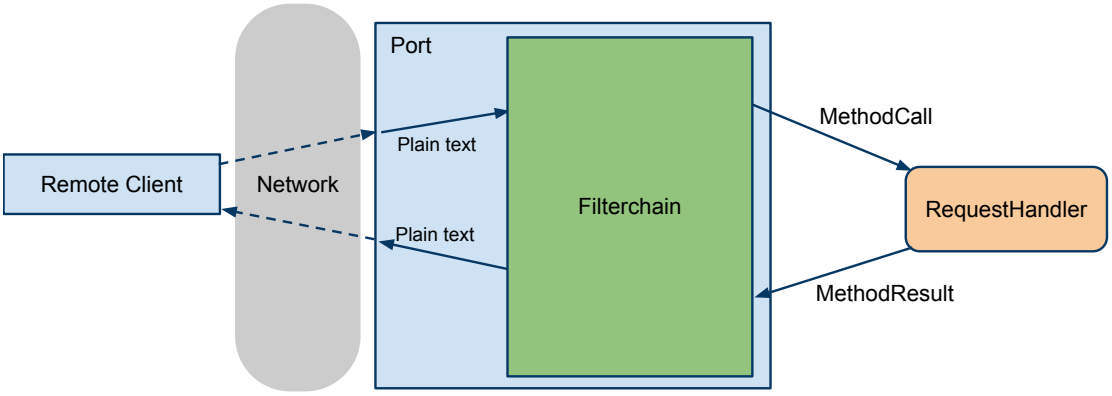

Figure 6.25: Filters in architecture

XML, only the marshalling filters must be exchanged, while the verification and authentication filters can be reused entirely. The encryption filter could also be reused. For XML messages however it would be more appropriate to use XML Encryption standard as defined by W3C.

A filter chain is part of the configuration of a port, and thus tied to a specific instance of a port (see Figure 6.25). So it is possible to provide a port in secure and unsecure configurations. Figure 6.26 shows a filter configuration with all security relevant filters in place. An similar

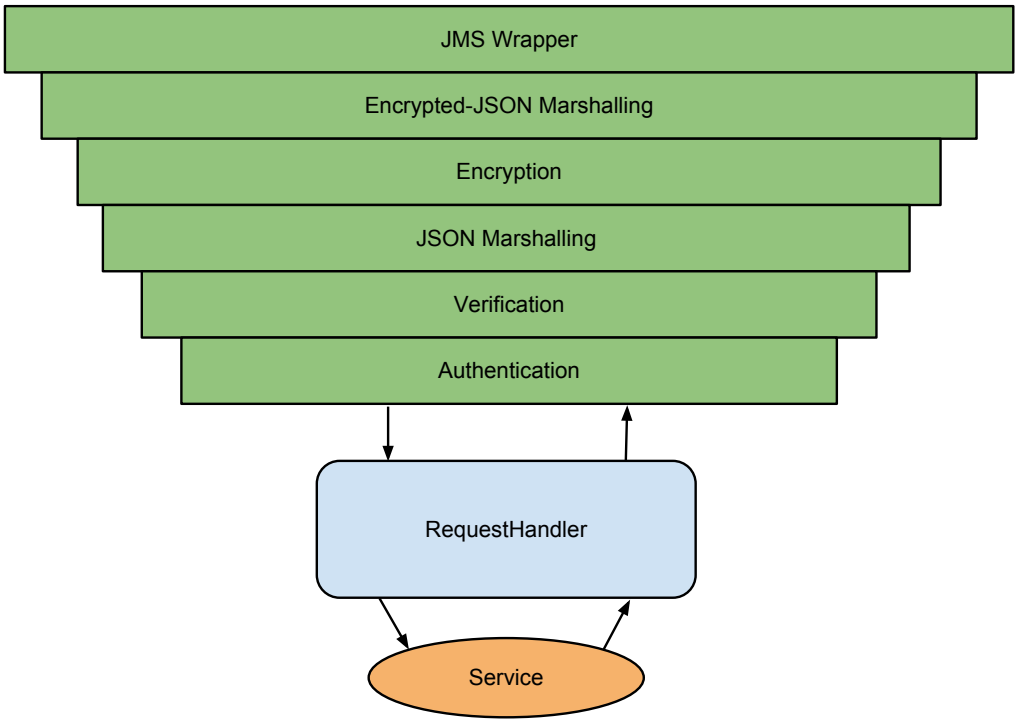

Figure 6.26: Filter configuration for a secure port

configuration without any security can be achieved by using a different list of filters (see Figure 6.27)). It is very common to provide a subset of functionality publicly without requiring authentication (e.g. generic contact information or the server's public key). Such filter configuration would not contain any elements for encryption, authentication or verification because it's not necessary. It is also very useful during development and for testing purposes to interact through unsecure ports.

6.6 Impact on Development

Security in an architecture imposes additional challenges to engineers developing integration solutions based on it. As described in section 2.2 different points of development are considered:

Attaching the Security Interceptor

The most important component of the security concept is the security broker in the form of a method interceptor. It must be attached to all security sensitive services. So when developing core components it must be made sure that the interceptor is attached to the registered service. When services are registered without an interceptor they can be discovered and invoked by other components. Attaching the interceptor to all services is not an option, since some services are

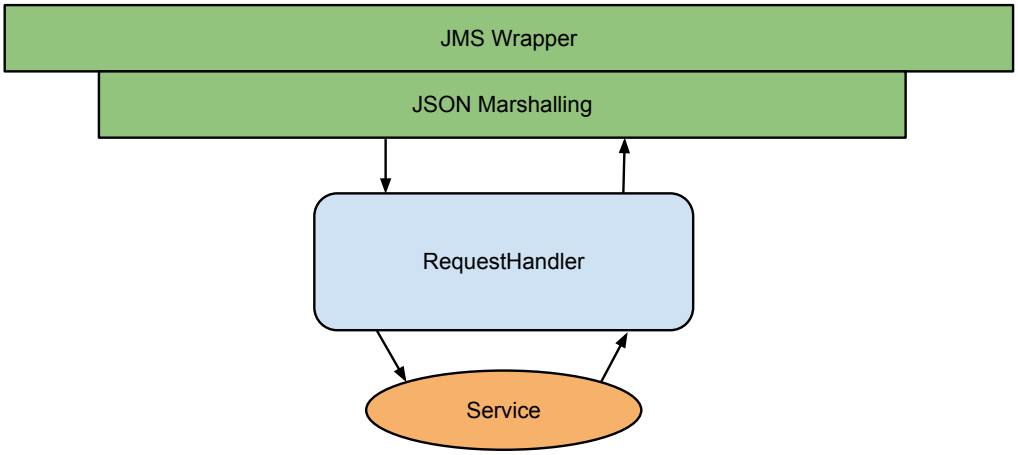

Figure 6.27: Filter configuration for a port without security

supposed to be registered without it. Developers of domains and connectors do not need to take this into account, because the Connector Manager service takes care of securing connector instances.

The Security Context

In the presented solution the security context is stored in a thread local variable using mechanisms implemented Apache Shiro. Using thread locals bears certain risks.

The access to the thread local variable is not controlled. This means developers must be careful when modifying the content of the variable, because it affects every action performed in this thread. There are very few cases the developer is required to modify the contents of the security context directly though. Storing the authentication token after successful authentication is handled by authentication connector composition strategy. To execute actions with elevated privileges the developer is required to encapsulate the action into a task that can be executed in a sandbox. However bad code in an active bundle can render the Security Interceptor completely ineffective.

Another issue that comes naturally when using thread local variables, is what happens when performing an action requires concurrent execution using threads. In Shiro, the new thread inherits the security context from the parent thread. The context is then shared by all these threads, meaning when the authentication becomes invalid, it becomes invalid for all threads.

Using thread locals is always dangerous when using thread pools. For example when some user executes an action in a connector that uses a thread pool, the authentication of that user is stored in the thread spawned by thread pool. When another user now invokes an action on the same connector, the thread from the previous action is reused with the previously stored authentication. So the second action may be executed with privileges of the user that performed the first action.

So special thread pools are required, that take the security context into account when accept-

ing new tasks. Bundles that use regular thread pools disrupt the authorization process, because decisions are based on the wrong users.

So developers must consider the following when developing a bundle that is deployed along with the OpenEngSB:

- use caution when modifying the security context
- use caution when spawning new threads, especially with threadpools

Remote Security

By default, remote security is handled with the message encryption mechanism described in section 6.5. A plain remote service request without any security must be transformed into a secure request.

The first step is to include authentication information to the message. This has little effect on connector implementations since it is only a change of the message model. However it also imposes the restriction that the host OpenEngSB must be able to authenticate the user and grant the necessary permissions.

In addition the message must be encrypted. For this purpose the public key of the host OpenEngSB must be obtained. The challenge is to do it securely, because no secure channel with the OpenEngSB can be established without the public key. A simple solution is to transfer the key using external storage devices like an USB flash drive.

To implement the encryption process itself, the language must provide encryption implementations for the used algorithms. Implementations for common algorithms like AES and RSA are available in many languages, but the encryption of the message must be implemented by the developer.

So developers of components that interact with the OpenEngSB remotely (e.g. remote connectors) must consider the following additional requirements:

- Authentication and authorization are required for all actions
- Authentication information must be added to all messages
- Messages must be encrypted with algorithms supported by the host OpenEngSB
- The public key of the host OpenEngSB must be transfered to the remote client securely.

Custom Remote Security

The port infrastructure of the OpenEngSB is built on modular and dynamically exchangeable layers called filters. The message encryption and other security features are implemented as such filters, and can be exchanged with protocol specific mechanisms, like for example encrypting the communication channel with TLS [17]. Exchanging filters in the port configuration bares the risk of introducing insecure filters, that may expose sensitive information because of insufficient confidentiality or integrity.

It is also possible to configure a filter-chain that does not use or support any security at all. The security of the port configuration is not validated, so if the configuration is compromised by an unsecure layer, there is no automatic detection mechanism.. In some cases it may be desirable to use unsecure ports to provide less sensitive information to a wider range of consumers. However it must be made sure, that no sensitive information is returned through this port.

So when developers integrate custom security layers, the following must be considered for these implementations:

- When replacing secure filters, the replacement must provide the same security attributes as the original.
- When providing an unsecure port, make sure no confidential information is transmitted, because it could be intercepted and read by an attacker.

CHAPTER 7

Evaluation

All previous steps are part of the basic security concept which is the prime goal of this thesis. In order to validate the concept usecases were designed in chapter 4. First the basic usecases, designed to validate certain elements are considered.

7.1 Authentication of an External Connector

The test scenario is to install an example domain in a running openengsb and register a remote example connector with it. The definition of the example domain is shown in Figure 7.1.

The result of the log operation should be a copy of the output produced by the connector. In addition the implementation should raise an Event of the type “LogEvent”. The remote connector is implemented as a stand alone Java program.

The execution of the scenario is described in a way, an administrator would set it up. An integration test has been implemented that executes the same steps automatically.

After the OpenEngSB has been started the next step is to create a new user that the remote connector will use to authenticate. A user named “example” is created with a strong random generated password. This can be done by using the provided administration web interface (see Figure 7.2). In the integration test, the user management service is used directly.

In order to add support for exchanging JMS messages, the corresponding port must also be installed. This is done in the command shell using the provided features mechanism, which are

```
public interface ExampleDomain extends Domain {
    String log(String message);
}
```

Figure 7.1: Example domain definition

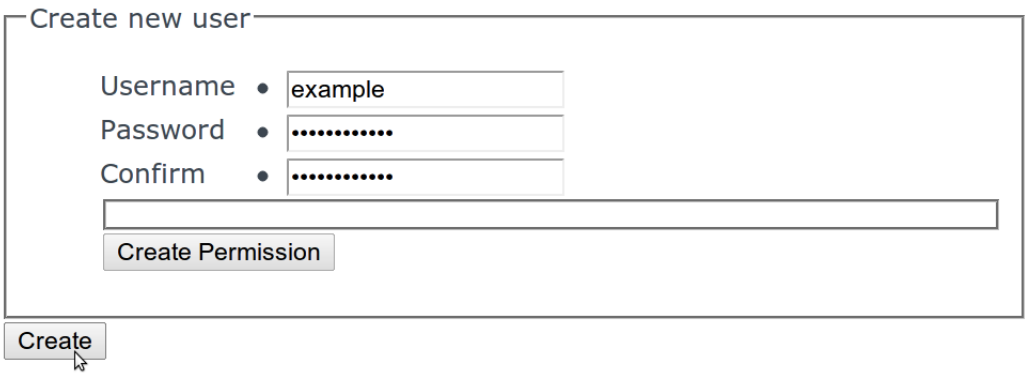

Figure 7.2: Create the user “example”

```
features:install openengsb-port-jms
```

Figure 7.3: Install the feature for JMS support

both provided by Apache Karaf[1]. In the integration test, the framework is configured with an additional config which features to load, that includes the jms port.

The feature is preconfigured with a outgoing port service named “jms-json”, that uses the secure filter-chain shown in Figure 6.27. The encryption filters are configured to use AES for encrypting messages, and RSA for encrypting the session keys. The algorithms are easily exchangeable so that the implementation of the protocol remains independent of the specific algorithm. The OpenEngSB is now ready to receive encrypted messages on the JMS queue named “receive”.

The first step when preparing the remote connector, is to setup the public key of the OpenEngSB. When the OpenEngSB is started for the first time, it generates a keypair and writes it to the configuration directory. Before the remote connector can be started it must be provided the public key of the host OpenEngSB. This is done by manually copying the file containing the public key to the remote machine. This is not done via the network, but using a physical storage device. So the key can be assumed to be trusted. In the integration test the remote client is started by the test container. It spawns a new process on the same machine.

The client connector can now be started with the OpenEngSB’s URL as startup parameter. When the connector is started, it needs to register with the OpenEngSB first, in order to be able to receive messages. This is done by calling the connector registration service (see Figure 6.22). Therefore it generates a session key with 128 bit length for use with the AES. Longer keys are possible but may not be supported in all versions of the Java Virtual Machine right away. The client will now invoke the OpenEngSB’s connector registry service by sending a JMS message. The message contains the session-key in Base64 encoded form and the credentials for the user

[1] http://karaf.apache.org/

```
{
  "principal" : "example",
  "credentials" : {
    "type" : "password",
    "data" : {
      "value" : "fqWNTQxZMkVB"
    },
  },
  "timestamp" : 1318256089032
  "message" : {
    "methodCall" : {
      "classes" : [ "java.lang.String", "[B", "java.lang.String" ],
      "methodName" : "registerConnector",
      "metaData" : {
        "serviceId" : "connectorRegistry"
      },
      "args" : [  "example-remote",
                  "K26r85ddhf0tLtX1u3mAFztVf7QHxAkm/7YPOtRcE7s=",
                  "AES" ]
    },
    "answer" : true
  },
}
```

Figure 7.4: Message to register a remote connector with OpenEngSB

```
{
  "message" : "9pa/tb6gfD[...]ICHe6f63kP",
  "key" : "L+wcsjLixS[...]I9ff8buA=="
}
```

Figure 7.5: Encrypted form of the registration message

previously created for the connector.

This message is then encrypted using either the generated session key or a newly generated key. The session key used for encrypting the message is then encrypted using the configured public key. The result is then wrapped into a JSON message to delimit the message from the encrypted key (7.5).

The message is then transported via JMS as a text message. The OpenEngSB decrypts the message and processes it through the filter chain. Since there is no connector with the same name registered, the registration is always successful. The key is stored in a database and used for any future outgoing invocation to the connector.

As soon as the connector is registered, it is available for creation of a remote connector proxy. This is done by deploying a configuration file in the OpenEngSB (Figure 7.6. It is also

```
portId=jms-json
destination=tcp://192.168.1.1:6549?queue=example-remote
```

Figure 7.6: Configuration file for remote connector

```
public interface ScmDomain extends Domain {
    byte[] get(String path);
}
```

Figure 7.7: Part of the definition of the SCM domain

possible to use the administration interface for that.

The file is named "example+proxy+example-remote.connector" so that the deploy service recognizes the domain and connector type. The deployer creates a service in the OpenEngSB that handles invocations to the remote connector transparently. The remote communication is transparent to workflows and other components using the connector. It is also protected by the security broker. The setup for two way communication between the OpenEngSB and the remote connector is now finished. This means the connector can be invoked by the host OpenEngSB and raise events.

The connector can now be called using the Testclient interface of the OpenEngSB. In the integration test the proxy connector is discovered by using the appropriate OSGi-filter. The result of the call should be equal to the argument entered. To check whether the event has been raised, the OpenEngSB is preconfigured with a connector implementing the "auditing domain". It intercepts all events and stores them. So calling the "getAllAudits" in the corresponding connector should return a list containing a LogEvent containing an attribute with the entered message as value. In the integration-test the auditing-domain is queried directly by resolving the service in the OSGi-registry.

7.2 Hierarchical Policy Management

The next usecase shows that the concept can support hierarchical access control. The Source Code Management domain with a reference implementation for the Git SCM is used for this. The domain provides the method "get" and returns the content of the file at the given location inside the repository (see Figure 7.7). The actual definition of the domain contains a number of additional methods that are not important for this scenario. An instance of a Git connector is configured with the path of the repository containing the data.

Because the Git connector and the SCM domain are developed as third party bundles in dedicated projects, the automatic integration test uses the previously described example domain for validation.

First two users "Alice" and "Bob", and the roles "CEO" and "engineer" are created using the administration interface. In the integration test the user manager is used directly as a service. Two contexts "P1" and "P2" are created to represent the projects. Now connectors are configured

```
attribute.repository="/var/repositories/P1"
attribute.branch="master"
property.location.P1="scm/main"
```

Figure 7.8: Git Connector Configuration

Figure 7.9: Successful invocation in project P1

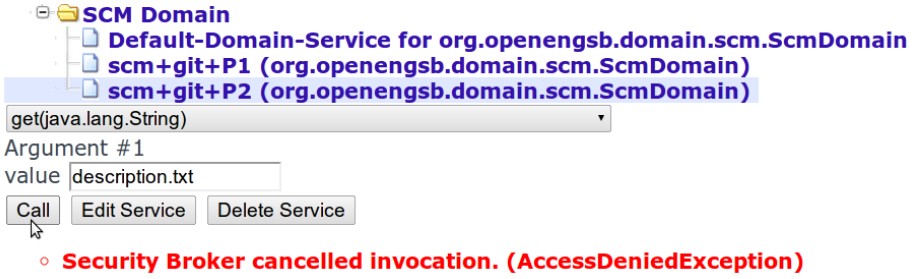

Figure 7.10: Failed invocation in project P2

for each project, by deploying configuration files.

A service permission allowing full access to all services in the SCM domain is added to the role CEO. The engineer role is assigned a similar permission but restricted to the connector instance created for “P1”. Alice is assigned the engineer role and bob the CEO role.

To execute the test scenario the user authenticates as “Alice” and switches to the context “P1”. Then the test client in the administration interface is used to call the “get” operation on the connector “scm+git+P1”. The result is the content of the file (see Figure 7.9).

When trying to call the same operation for the same path on the instance “scm+git+P2”, an Exception is thrown (see Figure 7.9).

As the message indicates the security interceptor attached to the service that represents the connector did not allow the invocation to proceed because the user did not have the proper authorization.

When the user authenticates as “Bob”, switches to the context “P2”, and then tries to get the content of the project description the same way Alice just tried, the result is the content of the

```
@DomainAuthorization("scm-repository-file")
byte[] get(@PathParameter(access=READ) String file);
```

Figure 7.11: SCM domain with domain specific security

description in P2.

As described in section 6.3 the security interceptor uses an attached authorization connector to determine whether the user should be granted access. The connector relevant in this context is the Service Permission Connector. With each invocation the user management is queried for the permissions the user has been granted. The user management makes sure, that all roles are traversed and all permissions and implied permissions are returned accordingly. The connector than makes a decision based on the set of granted permissions and the ongoing method invocation. When Alice is authenticated the user management only returns the service permission for the instance "scm+git+P1", because it is granted by the "engineer" role. Because the service permission connector did not decide to grant access, and neither did other connectors, the security interceptor cancels the invocation before it reaches the connector itself.

Since these permissions were assigned to the users by assigning them roles, this proves that the role based access control implementation satisfies the requirements defined for RBAC by Ferraiolo et al [24]. Though not covered in this usecase, the user management also supports role hierarchies, by adding roles to other roles. So the full hierarchical RBAC specification is satisfied.

7.3 Fine granular Access Control

While the previous scenario only requires access control on service instance level, more fine grained access control is required now. This usecase covers access control on parameter level. As for the previous scenario the SCM domain is used. Only the "get" operation is important for this usecase. Instead of controlling access to the service instance itself, now the parameter is the factor the authorization decision is based on.

As described in section 5.4, this fine grained access control definition is attached to the definition of the domain. Integration into the security broker is achieved by registering services implementing the "FineAccessControl" interface. To implement the usecase the SCM domain is required to control access on parameters that denote a path, and decide about granting access depending on the parameters value. A service named "scm-file-access" is created and is responsible for making the access control decisions based on special file access permissions. This is done by examining the method invocation.

As described in 6.3 every method that requires domain specific authorization is annotated with the DomainAuthorization annotation. To help the "scm-file-access"-service to determine which parameter represents a path, the parameters are annotated with an annotation for that (see Figure 7.11).

A file access permission used by the access control service to make a decision defines the following attributes:

- Path pattern. A pattern describing which paths the permission is granted on.
- Access type. Whether the permission allows read or write access on the file.
- grant. Whether to grant or deny access on the described pattern and action.

The "scm-file-access" service reads all PathParameter annotations from the method that is to be invoked and checks each value for access by comparing it to the file access permissions granted the user.

To execute the test scenario with the new components a single Git repository is created, and the "internal" directory is created in the repository. Files are added to the repository and the internal directory accordingly.

After the OpenEngSB has been started, the users "Alice" and "Bob", and the context "P1" are created. A connector that points to the project's Git repository is configured by deploying a configuration file. A service permission for the created Git connector is added to both users, so it allows both users to execute the "get" operation.

7.4 Tool Integration

In order to get write access to a remote Git repository, the user must authenticate by initiating an SSH session [62]. Authentication can be performed using a password or public/private keypair. Many servers allow public key authentication only. In this scenario a Git connector is configured pointing to a remote repository. To perform write operations such as committing a new file to the repository, the connector must authenticate with the repository.

To do this, the connector must implement a public key authentication adapter. Before the connector can perform a write operation on the repository the authentication adapter must authenticate the user using a private key as credentials (e.g. an SSH-RSA key). The key is stored in the keychain of the user stored in the security context of the thread that invoked the connector.

If the action is not invoked by a user, but by for example a automatic background task, the connector must also have access to some credentials. For this purpose, the authentication for elevated privileges that is stored in the security context when executing such actions must also be associated with a credentials collection to obtain the private key from.

7.5 Signal Exchange

The previous test scenarios show that the fundamental usecases can be realized with the presented components. The "Signal Exchange" presents a practical application required in electrical engineering environments. The main goal is for two companies to seamlessly work together on specifications that each company handles with its own software tools. A simplified version of the signal domain is defined (see Figure 7.12).

In addition the domain defines the "SignalChangedEvent", that is raised by connectors when they detect a changed signal in the connected tool. In order for external connectors to trigger an event, it must be transported to the host OpenEngSB. Raising an event is equivalent to a

```
public interface SignalDomain extends Domain{
    void updateData(SignalList signals);
}
```

Figure 7.12: Signal Domain

```
public interface WorkflowService {
    void processEvent(Event event);
}
```

Figure 7.13: Workflow Service

service request to the workflow service. The workflow service is an interface defining methods for processing events (see Figure 7.13)

To raise an event, the remote connector must send a service request for the workflow service to the OpenEngSB. The connector adds authentication information to the request, encrypts it with a generated session key and then encrypts the session key with the public key of the host OpenEngSB (see Figure 7.14). Intercepting the encrypted message is of little value to a possible attacker as the original key cannot be recovered without knowledge of either temporary session key or the private key of the host OpenEngSB. The time stamp attached to the secured message protects the environment from replay attacks, by only allowing messages with newer timestamps. The time stamp is part of the encrypted message and thus cannot be altered without decrypting the message first.

When the OpenEngSB receives the message, it is decrypted and authenticated using the provided credentials. The event is then processed by the workflow engine which may apply the changes to the stored reference in the EDB, so that the next component that requests an update, gets the updated version. The workflow can also involve a step where some connectors are updated immediately after the change has been applied. This only works when the connectors are online.

Tools that are operating offline most of the time, need means to acquire a updated working copy from the host OpenEngSB. A "RequestUpdate" event is sent to the OpenEngSB as just described. The triggered rules and workflows may perform some preparations for the checkout operation and eventually provide the connector with the updated data.

The openengsb transports the updated data to the connector by calling its updateData operation. The method request is encrypted with the connector key registered with the OpenEngSB as described in Section 6.5. Because the message that is transported over the network is encrypted, the attacker cannot examine or modify the message. It is however the client connector's responsibility to check the timestamp used in the request.

So the signal list can be edited and exchanged securely by both sides both ways. The company on the client side of the workflow can either run its own OpenEngSB to collaborate with the other companies OpenEngSB, or use a lightweight client side java program to provide the link.

```
{
   "principal":"connector",
   "credentials":{
      "type":"password",
      "data":{
         "value":"password"
      },
   },
   "message":{
      "methodCall":{
         "classes":[
            "org.openengsb.core.api.Event"
         ],
         "methodName":"processEvent",
         "args":[
            {
               "name":"TestEvent",
               "type":"Event"
            }
         ],
         "metaData":{
           "serviceId":"workflowService"
         }
      },
   },
   "timestamp":1319030455965
}
```

Figure 7.14: Workflow Service Request

7.6 Continuous Integration

In the continuous integration scenario, the CI&T workflow is implemented using OpenEngSB domains and connectors. It is not important which connectors are used in the actual runtime environment, since the security concepts and configuration are independent of specific connector configurations.

The connector implementing the SCM domain, monitors the configured repository and raises an Event when a new commit is added to the repository. The workflow engine contains a rule, that in case of such an event starts the CI&T workflow. The three main steps “build”, “test” and “deploy” use domains to describe the actions that are to be taken (see Figure 7.15).

In a project, each of the domains is assigned a specific connector instance. Regular users do not have access to the connectors used for the steps of the workflow. Also a regular user may not start the workflow manually. This is enforced by the security interceptor that checks service permissions on the workflow service.

When the SCM connector raises the event, it may be triggered either by an authenticated

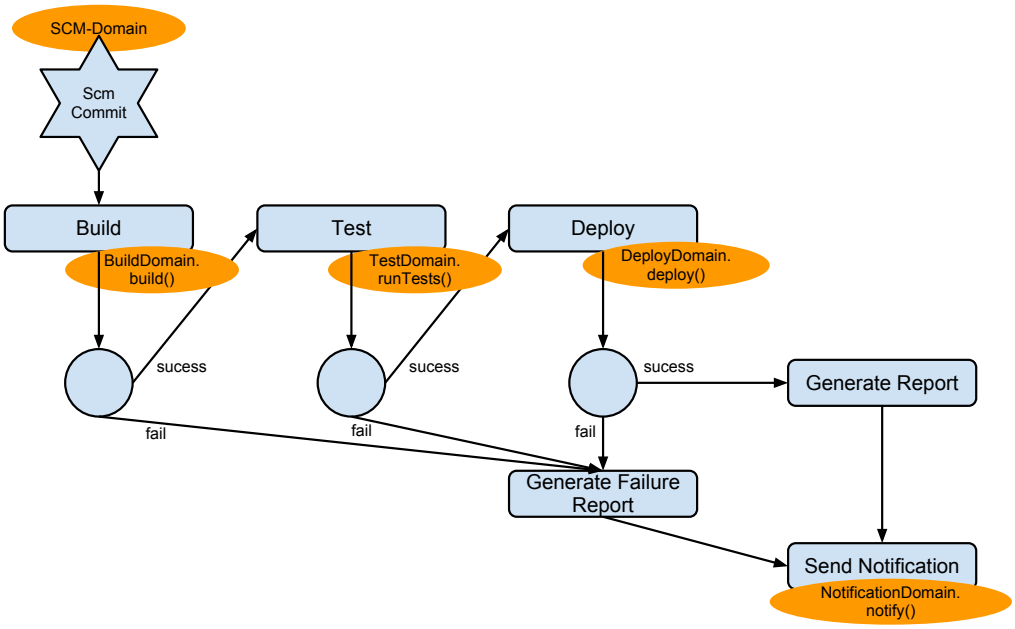

Figure 7.15: CIT workflow with domains in steps

OpenEngSB user, but also by a background timer that regularly checks the state of the repository. In the latter case, the connector must authenticate in order to raise an event. The connector implementation can use the mechanism for executing operations with elevated privileges as described in section 6.3. When the user intends to start the workflow manually, he needs the necessary service permissions.

Each domain referenced in the workflow may be represented by an external connector. If for example the step "test" is intended to be executed on a remote machine, a remote test connector must be implemented. Alternatively another OpenEngSB instance can be used to provide the connector. It must be running on the remote machine and is connected to the OpenEngSB by registering a remote connector as described in section 6.5. The connector authenticates with the OpenEngSB using encrypted messages, thus making the communication secure.

So the OpenEngSB is used to realize a distributed CI&T workflow. The presented concepts can be used to secure the execution and control over the workflow.

7.7 Impact on Performance

Adding security to the OpenEngSB and the external communication adds additional complexity to all involved components. Using the message encryption mechanism causes the messages to be greater in size, but also increase the processing cost. The security interceptor also adds additional cost because of the additional security assertions.

The two main performance indicators for message exchange is message size and processing time. The message size itself is an indicator for potentially greater network latency. Both depend

Parameter size	Throughput without security	Throughput with Security
1000	12.1 per second	9.9 per second
1000000	4.1 per second	2.5 per second

Table 7.1: Throughput depending on parameter size

Parameter size	Size of plain message	Size of encrypted message
1000	1308	2221
1000000	1000308	1334209

Table 7.2: Message size depending on parameter size

on the original size of the message. Therefore one short and one long message will be used for evaluation.

The messages are assumed to be encoded in JSON and encrypted using AES with key length of 256 bit for symmetric encryption, and RSA with a key length of 2048 bit. The following steps are performed for an incoming message:

- Unmarshal the encrypted message
- Decrypt the session key using the stored private key
- Decrypt the message using the session key
- Unmarshal the decrypted message
- Verify the message
- Perform authentication

When performing unsecure message processing, only the unmarshalling of the plain message is performed.

For the tests an invocation to an instance of the example connector is used. The argument of the method called, is returned by the connector implementation unmodified. In order to vary the size of the message, the length of the argument is modified. The two test messages use random strings of length 1000 and 1000000 characters.

The test was performed by sending messages continuously for one minute. After that the number of received responses is taken as measure to calculate how many messages can be processed in one second.

So adding security causes noticeable decrease in throughput with small messages. However in (Software+) Engineering environments artifacts are exchanged over the network quite often, making large messages very common. For large messages the processing time is increased significantly because the encryption and transformation is more expensive.

The size of the encrypted message is less significant for larger messages. So the transport overhead for large messages is quite low.

Parameter size	Throughput without security	Throughput with Security
1000	535.2 per second	296.5 per second
1000000	73.4 per second	37.2 per second

Table 7.3: Processing throughput depending on parameter size

In an additional test, only the processing time of the message is analyzed. In this test no OSGi-service lookup must be performed. Client side processing is excluded.

According to the results shown in table 7.3, the presented default implementation of the security layer decreases the processing speed of messages by up to 50 percent. The prototype implementation is not optimized for performance. Possibilities for improving performance may be switching encryption algorithm or implementation. Improved caching may avoid many unnecessary calculations. Alternatively replacing the message encryption with TLS when applicable may also improve performance.

CHAPTER 8

Discussion

In this section the findings from implementing the security broker solution are discussed in the context of the research issues defined in chapter 3. The basic goals are providing a basic security framework and the integration in multi-disciplinary tool environments. Also the consequences of the integration of security are discussed. At the end some limitations of the solution are described.

8.1 Authentication

The goal of RQ1 is to determine methods for authenticating parties in distributed tool integration environments.

The result are new components that allow remote entities to authenticate over arbitrary protocols and mechanisms. Best practices from web services that are also used in existing frameworks were used in the solution. The solution reuses the tool domain architecture that is used to integrate tools to integrate different authentication mechanisms.

In typical data integration approaches solutions are based on WS-Security [45] and WS-SecureConversation [44]. WS-Security specifies a specific XML based protocol to transport authenticated messages but is independent of specific authentication mechanisms. This means that all parties must agree to implement these standards.

In tool integration, the capabilities of the tools must be considered as well. Many tools especially in domains other than software engineering do not support any SOA related standard for communication and data exchange. Some tool suites (usually from the same vendor) use proprietary data formats to exchange data.

To minimize the integration effort, these proprietary formats should be reused ruling the use of web services out as a generic solution. The proposed solution is intended to be independent of authentication mechanisms as the WS-* standards are as well. In addition, it is also independent of specific data formats or protocols used. The integration effort can be reduced this way, because introducing a new protocol into the SOA-environment is supposed to be easier than implementing a new feature into third party tools.

So the proposed solution can be divided in two parts: the authentication process and the protocol.

Authentication Domain

The authentication process is a generic way of introducing and integrating new authentication mechanisms. This is solved using the tool domain concept proposed by Pieber [49]. The common concept of all authentication mechanisms is to verify the identity of a subject using a set of credentials. So the authentication domain defines methods to support arbitrary authentication mechanism implementations. The concept of composite connectors allows combining several authentication mechanism either as alternatives or multi-factor authentication. The prototype implementation provides only a strategy for combining authentication connectors that requires successful authentication with at least one connector. This is the default behavior of a number of security frameworks as well.

Authentication Protocol

The other challenge in authentication is to securely transport the authentication over the network and prevent an attacker from stealing credentials or manipulate the message to impose the user and gain access to the system. Two important entry points for interactions were identified: The web application and ports that are able to receive service requests (e.g. process an event).

In order to secure web applications the most reasonable approach is to rely on standard protocols and mechanisms like HTTPS [51]. HTTPS is supported by most of the browsers in use today and is therefore easy to deploy on new clients. It also mitigates many of the threats to confidentiality and integrity. Alternative approaches may require third party plugins for the browser (e.g. Convergence [1] based Perspectives Concept introduced by Wendlandt et al [61]) or dedicated third party software on every client. The effort for deployment increases when distributing development, since clients are not necessarily in the same network or even the same company anymore.

When integrating tools relying on standards is not always possible. Many tools implement only proprietary formats and protocols. Also a wide range of programming languages is used to implement them (Java, C#, C++, etc.) Recent versions of some tools support SOA and provide interfaces that rely on WS-standards (see section 2.3). These interfaces can easily be accessed from any programming language and thus makes integrating the tools easy. Since Web Services mostly rely on HTTP transports, confidentiality can be achieved by using HTTPS in some cases. When HTTPS is not an option because of problems mentioned by Nakamur et al [46], services rely on message encryption using the XML encryption standard [19].

However there are tools that do not expose functionality using standardised service interfaces and only provide access through tool specific binary protocols. To integrate those as well a modular port-infrastructure capable of supporting arbitrary communication protocols is used in the OpenEngSB. The system has been modularized further to provide reusable message processing steps. The security of the a port implementation itself is hard to assert automatically. An

[1] http://convergence.io/

implementation of a secure port is given in section 6.5. It uses JSON formatted messages and encrypts the messages with generated AES keys called session key. The session key is encrypted with the servers public key. The public key must be transported to the client securely before any secure communication can happen.

8.2 Authorization

RQ2 focuses on access control on data and services. The goal is to provide a framework to integrate access control independent of the tool that is secured and also independent of how the access control decisions are made.

The core of the result is the security interceptor that controls access to infrastructure and data. It uses the tool domain approach to integrate authorization decisions from multiple sources.

In data integration scenarios access to data is often controlled by the backing database management system or some intermediate software layer. There are certain access control schemes that are widely in use in todays applications (see section 2.9). Hierarchical Role based access control (RBAC) is one of the most common solutions to manage complex policy hierarchies.

In tool integration access to infrastructure must also be controlled. In web services XACML [43] has become the standard for defining access control policies [6]. XACML is based on attribute based access control (ABAC) which is an abstract concept and can be used for RBAC as well [1]. A concept for authorization is required that allows controlling access to data and infrastructure. In addition it should be independent of the source where the policy descriptions come from.

So the access control component is divided into describing permissions and enforcing them.

Authorization Domain

An authorization domain is introduced to provide a generic way for describing permissions. The domain defines a method for authorizing access to objects by a subject. Connectors may support certain types of objects (like method calls or database entries). No specific storage backend is enforced on the connectors. A connector may get the data for a decision from a database or an XACML file. For simplicity a default backend has been introduced in the form of the UserDataManager service. It is a generic storage for users, permissions and permission sets. All authorization connectors implemented in the prototype use this as the backend. The data is stored in a relational database using the JPA[2]. JPA allows to use in memory databases as well as dedicated database servers to use as data source. The implementation of the UserDataManager can easily be replaced by an alternative implementation (e.g. using LDAP), by registering a service with higher priority. The UserDataManager specifies methods for managing permission sets which can be seen as roles in RBAC. Permission sets may contain other permission sets making hierarchical RBAC possible. So the UserDataManager serves as a central service for administrating users and permissions in the environment.

This generic approach allows access control on class, service and method level. To support access control on the parameter level, the semantics of the parameters are important. Since these

[2]http://www.oracle.com/technetwork/java/javaee/tech/persistence-jsp-140049.html

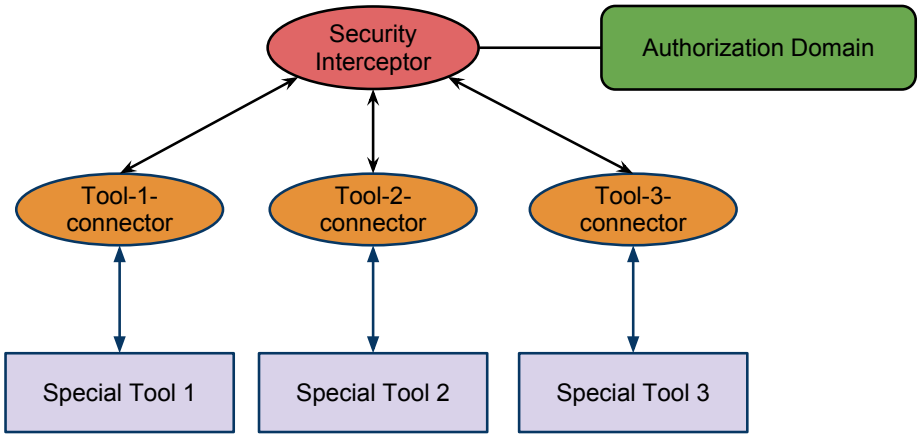

Figure 8.1: Unified access control with Authorization Domain and the Security Interceptor

semantics are specific to each domain, this is the place where special access control handling must be defined. So domains may specify additional access control handlers that control access on parameters.

The evaluation of the usecases shows, that the combination of the authentication domain with the appropriate connectors and the UserDataManager allows specifying fine grained permissions and managing them with project-specific and domain-specific roles.

When using alternative solutions to manage users and permissions (e.g. XACML), custom services and user interfaces must be implemented.

A great advantage of the presented solution is that managing access control policies for all tools can be centralized in one place (see section 8.1). So all permissions can be viewed in one place and be presented in a user-friendly way. In addition, the administrators are not required to learn tool specific access control mechanisms for every tool anymore.

Interceptor

To enforce access control on entities the requests are intercepted and checked for proper authentication and authorization on the resource.

The Java Servlet Technology[3] allows to add filters to a web application to intercept certain URL patterns. Using wildcards in these patterns allows all requests to be routed through a filter.

In the prototype implementation the Wicket[4] framework handles the filtering of the servlet requests. Wicket provides interfaces for placing custom access control interceptors before pages and components are rendered.

Access control to the business logic is handled by the interceptor responsible for securing services. The security interceptor is an aspect weaved into each service instance before it is registered in OSGi. This way all external and internal requests are routed through the intercep-

[3] http://download.oracle.com/otndocs/jcp/servlet-2.5-mrel-eval-oth-JSpec/
[4] http://wicket.apache.org

tor. The interceptor is responsible for authorizing requests using the configured authorization connector.

Forcing the security layer on internal service calls as well may lead to problems with background tasks performed by the system itself. To avoid this problem a concept for executing actions as a different subject has been introduced. It allows developers to execute certain actions as a system user that is granted all permissions. Granting all permissions is potentially dangerous because implementation errors in components using such logic may lead to elevation of privileges in unauthorized threads. For this reason these actions are forced into sandboxes, which makes such errors less likely.

8.3 Tools with Complex Security

The goal of RQ3 is to determine additional challenges introduced by tools with more complex security concepts.

The following additional challenges were identified:

- Tool specific security must be implemented in the tool connector (e.g. SSL-encryption)
- Components providing additional authentication mechanisms might be required (e.g. Kerberos, SSH-public-key).
- Connector implementations must authenticate with the tool. There are several ways to handle credentials in this context

Many tools used in engineering include some security features themselves and require authentication from a user (e.g. Jira[5], Git[6]). In many projects Git is used with SSH-publickey-authentication (see section 2.6).

To integrate tool specific security concepts in a (Software+) engineering environment, a solution has been outlined in section 5.5 but not implemented in the prototype. To support authentication with a remote tool, the tool connector must contain an adapter implementation, i.e. the connector implementation is responsible for performing the authentication.

Two approaches on where to retrieve the credentials from are considered in this thesis. One is to place a single set of credentials in the connector's configuration. All users that are authorized to use the connector, will authenticate to the remote tool with the stored credentials. The other approach is to use assign credentials to each user and store them in a keychain on the server. The user must authenticate with the OpenEngSB in order to gain access to its own credentials store.

Which approach to pursue depends on security requirements. If it is important for the remote tool to maintain its own log of user's accessing data, it is more appropriate to use the credentials store approach.

Possible implementations of both approaches are outlined in section 7.4. The usecase shows how to integrate a remote Git repository using the SSH public key authentication mechanism

[5] https://www.atlassian.com/software/jira

[6] http://git-scm.com/

(see section 2.6 through the already implemented Git connector. The mechanism requires an SSH-RSA private key on the connector's side for authentication.

The current implementation in the OpenEngSB relies on a key file located in the home directory of the user running the OpenEngSB. This means that all configured internal connectors access the same file to retrieve the necessary credentials to authenticate with the remote repository. So all connector instances can only use the same globally defined private key for authentication.

8.4 Impact of Security

The goal of RQ4 was to determine the impact of the security concepts introduced in this thesis. More specifically the consequences for developers of new components and administrators of the runtime system are discussed.

Developers of new components must keep security in mind in order to be able to interact with other components. Additionally security mechanisms must be included into the component to not compromise the security of the whole system.

For administrators the secure version of the runtime systems requires additional deployment effort. Also users and access control policies must be maintained. The solution allows existing infrastructure in this context (e.g. LDAP) to be integrated and reused.

Developers

Developers of external connectors must be able to use encryption and signing algorithms in the client implementation. The security could be abstracted by providing language specific bridge libraries. Such a bridge library is in development in a subproject of the OpenEngSB called Loom[7]. Currently only a C# library is supported, but it does not include an abstraction for the security layer yet.

The security layer is built on a modular architecture that allows exchanging certain aspects of security by alternative implementations. For example the message encryption could be replaced by a module that negotiates an SSL session with the remote host instead. There is no automatic assertion mechanism checking whether the replacement of such a module is equally secure. It is not feasible and sometimes not even desirable.

Administrators

Finally the security layer causes additional effort when deploying servers and clients because some additional security setup is required. Clients must be supplied with public keys of remote OpenEngSB instances. When using an SSL module in the implementation the certificate must be deployed as well.

8.5 Limitations

In this section some known limitations and shortcomings of the presented results are discussed.

[7] https://github.com/openengsb/loom-csharp-visualstudio

Performance Impact

The benchmark results shown in section 7.7 show the impact of introducing security on the performance of the system. A noticeable decrease in performance could be measured. It shows that the message processing may take up to 50% longer. However good performance was not a primary goal of the implementation. Thus many measures that would have improved performance have been omitted. There is no caching of authentications and authorization decisions. There is just some implicit caching provided by the JPA implementation. Replacing the message encryption with SSL when applicable could improve performance as well. García et al describe the impact of deploying SSL in Business to Business processes "can be quantified in a value between 5% and 10%" [29].

The impact on the message size is only significant for very small messages. So the network performance only decreases when exchanging small messages.

Offline scenarios

In Software engineering the development takes place in offices or sometimes at home. In either case, Internet connection can be assumed to be always available, so that the developer can access any artifact on the server. In other domains (e.g. electrical engineering or automation engineering) employees are sometimes required to visit a construction site where network access is often not available. However the employee might require access to certain artifacts or services provided only by the server. So local copies and caches need to be implemented on the client side. Replicating data for offline usage is critical to security because this data may contain company secrets. So an offline user should only get a small portion of the project's stored artifacts. But the artifacts may have dependencies to other artifacts in the same context. These dependencies are subject to change, and some of the changes have an actual impact on the artifact itself. Also they need to be available offline, while the whole secret plan is not.

The concept and prototype implementation presented in this thesis do not offer dedicated handling for offline replication as it requires deep knowledge of the semantics of the stored data. The concept however allows domains and connectors to be designed to exchange data and store it locally for offline usage. However there are additional challenges when considering artifacts with complex linking relations. A linked artifact may not be accessible to the user, while a certain linked part however should be. The approach for domain-specific parameter-based security shown in section 5.2 is suitable for tackling this issues when connected to the server, but caching such links locally is much more difficult.

CHAPTER 9

Conclusion and Future Work

In this section the results from this thesis are summarized. Section 9.1 summarized the findings and results of the thesis. Section 9.2 loose ends in the solution are pointed out and challenges for future work are proposed.

9.1 Conclusion

When integrating software tools in workflows with the goal to overcome domain and company boundaries, securing the interactions of the participating tools and components becomes mandatory. Current integration solutions like the Enterprise Service Bus (ESB) concept are based on Service Oriented Architecture (SOA). Integration of tools is realized using tool specific connectors. In order to integrate a service over multiple companies, they expose some of the functionality as web services to be accessed via HTTP or HTTPS using SOAP XML messages. In order to authenticate WS-Security, WS-Trust and WS-SecureConversation provide specifications on how to securely exchange credentials and establish secure communication channels when dealing with web services. Integrity and confidentiality is achieved by either encrypting the transport channel with SSL [17] However, it is required that the tool's functionality is exposed as a web service. This can be undesirable because of increased implementation effort or performance issues as Web services require the use of XML.

The proposed solution is based on the OpenEngSB concept introduced by Pieber [49] and Biffl et al [10] and is supposed to be more flexible than classic web service approaches. The prototype implementation is targeted to be a part of the OpenEngSB Open Source project [1]. It is designed to be deployed to OSGi-containers [58]. Through the use of a modular gateway infrastructure, it is possible to support arbitrary protocols to access an OpenEngSB based integration solution. The gateway infrastructure has been extended to support additional modules to provide confidentiality, integrity, and authentication. A prototype implementation was given that secures

[1] http://openengsb.org

the transport of JSON-formatted messages by encrypting them similar to practices used in the web service standards WS-Security [45] and XML-Encryption [19].

The authentication module delegates the authentication to a modular system based on the tool domain architecture presented by Pieber [49]. New authentication connectors can be added dynamically at any time and combined to multi-factor authentication systems. A prototype implementation is presented that implements a simple password based authentication process.

To perform access control in a (software+) engineering environment, current solutions delegate the authorization decisions either to the operating system or the relational database management system or the tool itself. This leads to fragmentation of access control policies that are hard to maintain. Administrating these policies requires knowledge of all corresponding tools' access control system (e.g. UNIX-based operating systems and SQL-based database systems)

Therefore, the solution developed controls all tool access and employs a new generic and unified access control system using the tool domain and connector architecture. As all tool connectors are represented as services in OSGi's service registry, the access to these services is intercepted by the so called "security interceptor" or "security broker".

The security interceptor forwards each request to a combination of connectors in the authorization domain. One connector is responsible for controlling access to services. It maintains the permissions for the users in a central permission repository. The repository allows permissions to be aggregated as roles and assigned to specific users. The presented prototype implementation supports hierarchical Role Based Access Control as specified in the proposed NIST standard by Ferraiolo et al [24].

These authentications are sometimes more complex than simple password-based systems (e.g. Git[2] with SSH public key authentication [62]). Two approaches for managing the required credentials were presented. One is to attach them to the connectors configuration, so that every subject invoking the service authenticates as the same user. This is enough for most implementations since authorization is handled by the Security Interceptor anyway. An alternative would be to assign each user its own set of credentials and store them securely on the server like a keychain. This keychain mechanism is not implemented in the prototype, but is considered an issue for future work.

When controlling only generic services with the security interceptor, it is hard to perform assertions on the content of the request (e.g. which file is affected). Therefore the security interceptor allows extensions to domain specifications to describe domain specific authorization mechanisms. These extensions can be integrated into the centralized permission repository if appropriate. In section 6.3 an implementation of authorization system specific to Source Code Management systems that controls access to files and directories is described. The advantage in this case is, that the same access control policies can be applied to a Subversion[3] as well as a Git repository.

Some of the consequences of introducing these security features into a (software+) engineering environment were outlined in section 6.6. The development of new internal and external components requires additional expertise in security and cryptography. Performance tests on the prototype implementation showed a 20 to 40 % decrease in performance with the additional

[2] http://git-scm.com/

[3] http://subversion.tigris.org/

security in place. The results indicate that the message encryption is responsible for most of the decrease.

9.2 Future Work

The prototype implementation presented in this thesis provides just a small set of implemented protocols and authentication mechanisms. Additional connectors supporting LDAP and Kerberos tickets should be provided in the future, since these systems are in use by many companies and provided in many ESB-based solutions too.

As previously stated, the solution causes a significant decrease in performance. Additional and more thorough performance testing should be done in the future. The solution should be completely analyzed using profiling techniques. More sophisticated caching mechanisms and alternative protocols or algorithms may also improve performance.

The solution is supposed to provide a reasonable degree of security, but has not been measured. It should be measured and appraised to be usable in cost and benefit calculations related to the degree of security. This is related to more performance tests to determine the costs of each module. It might be a good solution to replace the message encryption with TLS encryption. Skipping the encryption at all is an option if the consequences of information disclosure are not severe.

The metrics obtained can be used as Quality of Service (QoS) attributes, to enforce a certain security level on services that require it. More detailed security related QoS are specified in WS-SecurityPolicy [39]. This standard could be used to enhance the service registry with security aspects.

Bibliography

[1] Anne Anderson. Core and hierarchical role based access control (RBAC) profile of XACML v2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-rbac-profile1-spec-os.pdf, February 2005.

[2] Anne Anderson. Web Services Profile of XACML (WS-XACML) Version 1.0. http://www.oasis-open.org/committees/download.php/24951/xacml-3.0-profile-webservices-spec-v1-wd-10-en.pdf, August 2007.

[3] Anne Anderson and Hal Lockhart. Profiles for the OASIS Security Assertion Markup Language (SAML) V2.0. http://docs.oasis-open.org/security/saml/v2.0/saml-profiles-2.0-os.pdf, February 2005.

[4] Anne Anderson and Hal Lockhart. SAML 2.0 profile of XACML v2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-saml-profile-spec-os.pdf, February 2005.

[5] Alessandro Armando, Roberto Carbone, Luca Compagna, Jorge Cuellar, and Llanos Tobarra. Formal analysis of saml 2.0 web browser single sign-on: breaking the saml-based single sign-on for google apps. In *Proceedings of the 6th ACM workshop on Formal methods in security engineering*, FMSE '08, pages 1–10, New York, NY, USA, 2008. ACM.

[6] E. Bertino, L. Martino, F. Paci, and A. Squicciarini. *Security for Web Services and Service-Oriented Architectures*. Springer, 2009.

[7] Elisa Bertino, Barbara Catania, Elena Ferrari, and Paolo Perlasca. A logical framework for reasoning about access control models. *ACM Trans. Inf. Syst. Secur.*, 6:71–127, February 2003.

[8] Elisa Bertino and Ravi Sandhu. Database security-concepts, approaches, and challenges. *IEEE Trans. Dependable Secur. Comput.*, 2:2–19, January 2005.

[9] Abhilasha Bhargav-Spantzel, Anna C. Squicciarini, Shimon Modi, Matthew Young, Elisa Bertino, and Stephen J. Elliott. Privacy preserving multi-factor authentication with biometrics. *J. Comput. Secur.*, 15:529–560, October 2007.

[10] S. Biffl, A. Schatten, and A. Zoitl. Integration of heterogeneous engineering environments for the automation systems lifecycle. In *Industrial Informatics, 2009. INDIN 2009. 7th IEEE International Conference on*, pages 576 –581, june 2009.

[11] Stefan Biffl and Alois Zoitl. Evaluation report january 2010 to june 2011. Technical report, Christian Doppler Laboratory for Software Engineering Integration for Flexible Automation Systems, Vienna, August 2011.

[12] Ebrima N. Ceesay, Coimbatore Chandersekaran, and William R. Simpson. An authentication model for delegation, attribution and least privilege. In *Proceedings of the 3rd International Conference on PErvasive Technologies Related to Assistive Environments*, PETRA '10, pages 30:1–30:7, New York, NY, USA, 2010. ACM.

[13] Yuen-Yan Chan, Sebastian Fleissner, Joseph K. Liu, and Jin Li. Single sign-on and key establishment for ubiquitous smart environments. In Marina L. Gavrilova, Osvaldo Gervasi, Vipin Kumar, Chih Jeng Kenneth Tan, David Taniar, Antonio Laganà, Youngsong Mun, and Hyunseung Choo, editors, *ICCSA (4)*, volume 3983 of *Lecture Notes in Computer Science*, pages 406–415. Springer, 2006.

[14] Ya-Fen Chang, Chin-Chen Chang, and Jui-Yi Kuo. A secure one-time password authentication scheme using smart cards without limiting login times. *SIGOPS Oper. Syst. Rev.*, 38:80–90, October 2004.

[15] D.A. Chappell. *Enterprise service bus*. O'Reilly Series. O'Reilly, 2004.

[16] J. Daemen and V. Rijmen. AES Proposal: Rijndael.

[17] T. Dierks and E. Rescorla. The Transport Layer Security (TLS) Protocol Version 1.2. RFC 5246 (Proposed Standard), August 2008. Updated by RFCs 5746, 5878, 6176.

[18] Whitfield Diffie and Martin E. Hellman. Multiuser cryptographic techniques. In *AFIPS National Computer Conference*, pages 109–112, 1976.

[19] Donald E. Eastlake, Joseph M. Reagle, Takeshi Imamura, Blair Dillaway, and Ed Simon. Xml encryption syntax and processing. World Wide Web Consortium, Recommendation REC-xmlenc-core-20021210, December 2002.

[20] Carl Ellison and Bruce Schneier. Ten Risks of PKI: What You're Not Being Told About Public Key Infrastructure. *Computer Security Journal*, 16(1):1–7, 2000.

[21] Liang Fang, Samuel Meder, Olivier Chevassut, and Frank Siebenlist. Secure password-based authenticated key exchange for web services. In *Proceedings of the 2004 workshop on Secure web service*, SWS '04, pages 9–15, New York, NY, USA, 2004. ACM.

[22] Niels Ferguson, Bruce Schneier, and Tadayoshi Kohno. *Cryptography Engineering - Design Principles and Practical Applications*. Wiley, 2010.

[23] David Ferraiolo and Vijay Atluri. A meta model for access control: why is it needed and is it even possible to achieve? In *Proceedings of the 13th ACM symposium on Access control models and technologies*, SACMAT '08, pages 153–154, New York, NY, USA, 2008. ACM.

[24] David F. Ferraiolo, Ravi Sandhu, Serban Gavrila, D. Richard Kuhn, and Ramaswamy Chandramouli. Proposed nist standard for role-based access control. *ACM Transactions on Information and System Security*, 4:224–274, August 2001.

[25] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. Hypertext Transfer Protocol – HTTP/1.1. RFC 2616 (Draft Standard), June 1999. Updated by RFCs 2817, 5785, 6266.

[26] Terry Fleury, Jim Basney, and Von Welch. Single sign-on for java web start applications using myproxy. In *Proceedings of the 3rd ACM workshop on Secure web services*, SWS '06, pages 95–102, New York, NY, USA, 2006. ACM.

[27] J. Franks, P. Hallam-Baker, J. Hostetler, S. Lawrence, P. Leach, A. Luotonen, and L. Stewart. HTTP Authentication: Basic and Digest Access Authentication. RFC 2617 (Draft Standard), June 1999.

[28] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design patterns: elements of reusable object-oriented software*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 1995.

[29] Daniel F. García, Rodrigo García, Joaquín Entrialgo, Javier García, and Manuel García. Evaluation of the effect of ssl overhead in the performance of e-business servers operating in b2b scenarios. *Computer Communications*, 30:3063–3074, November 2007.

[30] Satoshi Hada and Hiroshi Maruyama. Session authentication protocol for web services. In *Proceedings of the 2002 Symposium on Applications and the Internet (SAINT) Workshops*, SAINT-W '02, pages 158 –165, Washington, DC, USA, 2002. IEEE Computer Society.

[31] Michael Hafner and Ruth Breu. *Security Engineering for Service-Oriented Architectures*. Springer, Berlin, 1st edition, 2010.

[32] N. Haller and R. Atkinson. On Internet Authentication. RFC 1704, October 1994.

[33] N. Haller, C. Metz, P. Nesser, and M. Straw. A One-Time Password System. RFC 2289 (Standard), February 1998.

[34] E. Hammer-Lahav. The OAuth 1.0 Protocol. RFC 5849, April 2010.

[35] Michael Hauf, Janek Schwarz, and Andreas Polze. Role-based security for configurable distributed control systems. In *Proceedings of the Sixth International Workshop on Object-Oriented Real-Time Dependable Systems (WORDS'01)*, WORDS '01, pages 111 –118, Washington, DC, USA, 2001. IEEE Computer Society.

[36] Shawn Hernan, Scott Lambert, Tomasz Ostwald, and Adam Shostack. Uncover Security Design Flaws Using The STRIDE Approach, 2006.

[37] E. Kasanen, K. Lukka, and A. Siitonen. The Constructive Approach in Management Accounting Research. *Journal of Management Accounting Research*, 5:241–264, 1993.

[38] Charlie Lai, Li Gong, Larry Koved, Anthony Nadalin, and Roland Schemers. User authentication and authorization in the java(tm) platform. In *Proceedings of the 15th Annual Computer Security Applications Conference*, ACSAC '99, pages 285 –290, Washington, DC, USA, 1999. IEEE Computer Society.

[39] Kelvin Lawrence and Chris Kaler. WS-SecurityPolicy 1.2. Technical report, February 2009.

[40] J. Linn. Generic Security Service Application Program Interface Version 2, Update 1. RFC 2743 (Proposed Standard), January 2000. Updated by RFC 5554.

[41] Shintaro Mizuno, Kohji Yamada, and Kenji Takahashi. Authentication using multiple communication channels. In *Proceedings of the 2005 workshop on Digital identity management*, DIM '05, pages 54–62, New York, NY, USA, 2005. ACM.

[42] Richard Monson-Haefel. *Enterprise JavaBeans (3rd Edition)*. O'Reilly, October 2001.

[43] Tim Moses. eXtensible Access Control Markup Language (XACML) Version 2.0. http://docs.oasis-open.org/xacml/2.0/access_control-xacml-2.0-core-spec-os.pdf, February 2005.

[44] Goodner M. Gudgin-M. Barbir A. Granqvist H. Nadalin, A. WS-SecureConversation 1.4, February 2009.

[45] Kaler C. Hallam-Baker P. Nadalin, A. and R. Monzillo. OASIS Web Services Security: SOAP Message Security 1.0 (WS-Security 2004), March 2004.

[46] Yuichi Nakamur, Satoshi Hada, and Ryo Neyama. Towards the integration of web services security on enterprise environments. In *Proceedings of the 2002 Symposium on Applications and the Internet (SAINT) Workshops*, SAINT-W '02, pages 166 –175, Washington, DC, USA, 2002. IEEE Computer Society.

[47] Roger M. Needham and Michael D. Schroeder. Using encryption for authentication in large networks of computers. *Communications of the ACM*, 21:993–999, December 1978.

[48] B. Clifford Neuman and Theodore Ts'o. Kerberos: An authentication service for computer networks. In *IEEE Communications*, pages 33–38. IEEE Computer Society, 1994.

[49] Andreas Pieber. Flexible Engineering Environment Integration for (Software+) Development Teams. Master's thesis, Vienna University of Technology, 2010.

[50] David Recordon and Drummond Reed. Openid 2.0: a platform for user-centric identity management. In *Proceedings of the second ACM workshop on Digital identity management*, DIM '06, pages 11–16, New York, NY, USA, 2006. ACM.

[51] E. Rescorla. HTTP Over TLS. RFC 2818, May 2000. Updated by RFC 5785.

[52] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Communications of the ACM*, 21:120–126, February 1978.

[53] R.S. Sandhu and P. Samarati. Access control: principle and practice. *Communications Magazine, IEEE*, 32(9):40 –48, sep 1994.

[54] Flavio O. Silva, Joao A. A. Pacheco, and Pedro F. Rosa. A web service authentication control system based on srp and saml. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '05, pages 507–514, Washington, DC, USA, 2005. IEEE Computer Society.

[55] C. Steel, R. Nagappan, and R. Lai. *Core security patterns: best practices and strategies for J2EE, Web services, and identity management*. Prentice Hall PTR core series. Prentice Hall PTR, 2005.

[56] San-Tsai Sun, Yazan Boshmaf, Kirstie Hawkey, and Konstantin Beznosov. A billion keys, but few locks: the crisis of web single sign-on. In *Proceedings of the 2010 workshop on New security paradigms*, NSPW '10, pages 61–72, New York, NY, USA, 2010. ACM.

[57] The OSGi Alliance. OSGi service platform service compendium, release 4.2. http://www.osgi.org/Specifications, 2009.

[58] The OSGi Alliance. OSGi service platform core specification, release 4.3. http://www.osgi.org/Specifications, 2011.

[59] S. Vinoski. Java Business Integration. *Internet Computing, IEEE*, 9(4):89–91, 2005.

[60] F. Waltersdorfer, T. Moser, A. Zoitl, and S. Biffl. Version management and conflict detection across heterogeneous engineering data models. In *Industrial Informatics (INDIN), 2010 8th IEEE International Conference on*, pages 928 –935, july 2010.

[61] Dan Wendlandt, David G. Andersen, and Adrian Perrig. Perspectives: improving ssh-style host authentication with multi-path probing. In *USENIX 2008 Annual Technical Conference on Annual Technical Conference*, pages 321–334, Berkeley, CA, USA, 2008. USENIX Association.

[62] T. Ylonen and C. Lonvick. The Secure Shell (SSH) Authentication Protocol. RFC 4252 (Proposed Standard), January 2006.

[63] Eric Yuan and Jin Tong. Attributed based access control (abac) for web services. In *Proceedings of the IEEE International Conference on Web Services*, ICWS '05, pages 561–569, Washington, DC, USA, 2005. IEEE Computer Society.

List of Figures