

Context-based Access Control

Thomas Groß
tgr@zurich.ibm.com
IBM Zurich Research Laboratory

Saarland University
Saarbrücken

Advisors:

Dr. Matthias Schunter
mts@zurich.ibm.com
IBM Zurich Research Laboratory

Dr. Birgit Pfitzmann
bpf@zurich.ibm.com
IBM Zurich Research Laboratory

Prof. Dr. Reinhard Wilhelm
wilhelm@cs.uni-sb.de
Chair for Programming Languages and Compiler Construction
Saarland University

(Version 03/31/2003)

Erklärung / Declaration

Hiermit erkläre ich, dass die vorliegende Arbeit von mir und nur unter Einbeziehung der aufgelisteten Literatur und Hilfsmittel erstellt wurde (siehe Literaturverzeichnis am Ende der Diplomarbeit).

Hereby I certify the originality of this work. All publications and material used are listed in the bibliography at the end of this thesis.

Zürich, 26. March 2003

Acknowledgments

Foremost, I want to thank my advisor Matthias Schunter for his support. The discussions with him about important design decisions were very valuable for the development of our entitlement service. His guidance and comments about content as well as structure and didactics of this thesis helped me a lot.

I would like to thank my co-advisor Birgit Pfitzmann for her guidance and support in the protocol part of this thesis. Her detailed comments were of great value for the formalization and analysis of the SAML Single Sign-on protocol.

I want to thank my co-advisor Reinhard Wilhelm for his support and the opportunity to write this thesis at the IBM Zurich Research Laboratory.

I also would like to thank Michael Waidner, the manager of the group for network security and cryptography at the IBM Zurich Research Lab. Without his help this thesis would not have been possible.

I appreciated the cooperation with the members of the IBM Zurich Research Lab. I want to thank in particular Günter Karjoth for the discussions about access control products and the design of the entitlement service. I would like to thank Roger Zimmermann for the inspiring discussions about software engineering and design principles. Furthermore, I want to thank Joy Algesheimer, Klaus Kursawe, and Reto Strobl for the helpful discussions about protocol design, cryptography, and practical security.

Last but not least, I want to thank my colleagues in other departments of IBM. I appreciated the efficient and nice cooperation with Brook Lovatt, Anthony Moran, Warwick Burrows, and Brian Turner. The team work with them was productive and fun. I also would like to thank Doug Davis, and Alfredo Da Silva for their support with the web service deployment.

This thesis has been partially supported by the IBM Zurich Research Lab, however, it represents the view of the author only.

Abstract

Protecting web-sites and applications against unauthorized access is one of the most important problems of the Internet. So-called access control products decide, whether a user is allowed to access a certain resource or not. The technique of Context-based Access Control achieves this by analysis of fine-grained attributes of an access requesting user and the context of such an access control decision. Currently Context-based Access Control is an emerging technology in the Internet area and a prerequisite of various state-of-the-art e-commerce applications. The most important players on the market of access control products currently include the Context-based Access Control functionality in their products. It is in the scope of this thesis to solve the related problem of dynamical attribute provision for such a product.

Many of the attributes used in Context-based Access Control have to be determined in the very moment of the access control decision, but an access control product cannot acquire them itself. Such attributes are for instance the current time, the age of an user or the status of a payment. Thus, a dynamic attribute provision on demand is crucial for these applications and with it the availability of protocols and services to do so. Given this requirement, we analyze and design protocols and services for the dynamic attribute provision of Context-based Access Control. We divide the thesis in two parts, one theoretical and one practical.

Part I is more theoretical. There, we consider a single sign-on protocol of the OASIS Security Assertion Markup Language (SAML)¹. This is an emerging protocol standard for the provision of assertions about a user's identity. The single sign-on protocol allows a user to login only at a source site he trusts, while this site confirms his identity to other sites.

We refine this standard into a protocol schema similar to that ones common for cryptographic protocols. We describe the protocol steps in detail and name security relevant assumptions explicitly. We provide an exact analysis for the SAML single sign-on, which is the only one known to the author for such a protocol standard.

We discovered different attacks on the SAML single sign-on protocol including man-in-the-middle attacks and such ones based on information leakage. We propose a repaired protocol that is resistant against the described vulnerabilities.

Part II is oriented practically. In this part, we design a service for dynamic attribute retrieval. This service externalizes the attribute provision functionality of access control products. We use Web Service technology to guarantee interoperability with various access control products. We integrate the service in one given product as proof of concept. We use different design patterns and the techniques of exemplary specification and automated module testing in order to provide reliability, robustness and modularity. The resulting service fulfills these quality requirements and is suitable as framework for most kinds of attribute provision protocols.

The results of this thesis show on the one hand the feasibility of exact security analyses for protocol standards like SAML single sign-on. We see on the other hand that the externalization of an attribute provision service is possible and useful for access control products.

¹<http://www.oasis-open.org>

Kurzzusammenfassung

Der Schutz von Web-Seiten und Anwendungen gegen unauthorisierten Zugriff ist eines der wichtigsten Probleme des Internets. Zugriffskontrollprodukte entscheiden, ob ein Benutzer auf eine bestimmte Ressource zugreifen darf oder nicht. Die Technik der kontextbasierten Zugriffskontrolle realisiert dies durch Analyse von feinkörnigen Attributen eines zugreifenden Benutzers und des Kontextes einer Zugriffskontrollentscheidung. Zur Zeit ist kontextbasierte Zugriffskontrolle eine sich rasch weiter entwickelnde Technologie im Bereich des Internets und Voraussetzung für verschiedenste aktuelle Anwendungen des elektronischen Handels. Die wichtigsten Firmen im Markt der Zugriffskontrollprodukte nehmen gerade die Funktionalität kontextbasierter Zugriffskontrolle in ihre Produkte auf. Im Rahmen dieser Arbeit lösen wir das hier auftretende Problem der dynamischen Bereitstellung von Attributen für ein solches Produkt.

Viele der in der kontextbasierten Zugriffskontrolle verwendeten Attribute müssen im Moment der Zugriffskontrollentscheidung ermittelt werden, können aber nicht vom Zugriffskontrollprodukt selbst bereitgestellt werden. Solche Attribute sind zum Beispiel die aktuelle Zeit, das Alter eines Benutzers oder der Status einer Zahlung. Daher ist eine dynamische Bereitstellung der Attribute wesentlich für diese Anwendungen und mit ihr das Vorhandensein von geeigneten Protokollen und Diensten. Diese Anforderung gegeben, analysieren und entwerfen wir Protokolle und Dienste für die dynamische Bereitstellung von Attributen der kontextbasierten Zugriffskontrolle. Wir unterteilen diese Arbeit dazu in zwei Teile, einen theoretischen und einen praktischen.

Teil I ist theoretisch orientiert. Hier betrachten wir ein Single Sign-on Protokoll der OASIS Security Assertion Markup Language (SAML)². Dies ist ein Protokoll Standard für die Bereitstellung von Zusicherungen über Benutzeridentitäten. Wir verfeinern diesen Standard zu einem Protokollschema, ähnlich zu solchen, die in kryptographischen Protokollen gebräuchlich sind. Wir beschreiben die Protokollschritte detailliert und benennen sicherheitsrelevante Annahmen explizit. Wir stellen eine exakte Analyse für das SAML Single Sign-on Protokoll bereit, welche die einzige dem Autor bekannte für einen solchen Standard ist.

Wir entdeckten verschiedene Angriffe auf das SAML Single Sign-on Protokoll, unter anderem Man-in-the-middle Angriffe und solche basierend auf unerlaubtem Informationsfluss. Wir schlagen ein repariertes Protokoll vor, welches gegen die beschriebenen Angriffe resistent ist.

Teil II ist praktisch orientiert. Hier entwerfen wir einen Dienst zum dynamischen Bereitstellen von Attributen. Dieser Dienst externalisiert die entsprechende Funktionalität von Zugriffskontrollprodukten. Wir verwenden Web Service Technologie, um die Zusammenarbeit mit verschiedensten Zugriffskontrollprodukten zu garantieren und integrieren den Dienst in ein gegebenes Produkt als Bestätigung unseres Konzeptes.

Wir verwenden verschiedene Entwurfsmuster sowie die Techniken exemplarischer Spezifikation und automatisierter Modultests, um Zuverlässigkeit, Robustheit und Modularität zu gewährleisten. Der resultierende Dienst erfüllt diese Qualitätskriterien und ist als Plattform für die meisten Arten von Protokollen zur Attributbereitstellung verwendbar.

Die Ergebnisse dieser Arbeit zeigen einerseits, dass exakte Sicherheitsanalysen von Protokollstandards wie SAML Single Sign-on machbar sind. Andererseits sehen wir, dass die Auslagerung eines Dienstes zur Bereitstellung von Attributen möglich und von Vorteil für Zugriffskontrollprodukte ist.

²<http://www.oasis-open.org>

Contents

1	Introduction	6
1.1	Problem Description	6
1.2	Scope of this Thesis	8
1.3	Outline	9
I	Protocol Analysis of the Security Assertion Markup Language	11
2	Browser-based Protocols	12
2.1	Introduction to SAML	13
2.1.1	Example of SAML Data Types and Messages	13
2.1.2	The SAML SSO Profile	16
2.2	Related Protocols and Projects	16
2.2.1	The BBAE Protocol	16
2.2.2	Liberty	19
2.2.3	Shibboleth	19
2.2.4	MS Passport	19
3	Security Analysis of SAML SSO	20
3.1	Introduction	20
3.2	Refinement of the SAML SSO Profile	21
3.2.1	Notation	23
3.2.2	Participating Parties	24
3.2.3	Protocol Interface	24
3.2.4	Trust Model	26
3.2.5	Protocol Schema	28
3.2.6	State Model for the Protocol	35
3.3	Assumptions	40
3.3.1	Assumptions about the Setup	40

3.3.2	Assumptions about the Protocol Run	40
3.4	Attacks on the Unmodified Profile	40
3.4.1	Connection Hijacking / Replay Attack	41
3.4.2	Man-in-the-Middle Attacks	42
3.4.3	HTTP Referer Attack	45
4	Repair of the SAML Single Sign-on Profile	47
4.1	Proposed Changes	47
4.2	Improved Profile	48
5	Conclusion and Future Work	54
II	Design of the DynAdiEntitlementService	55
6	Introduction	56
6.1	Environment	56
6.1.1	Access Control Products	56
6.1.2	Web-Access Enforcement Products	57
6.2	Related Literature	57
6.3	Related Protocols	58
6.4	Design Patterns Used	59
6.4.1	Singleton	60
6.4.2	Factory Method	60
6.4.3	Adapter	61
6.4.4	Strategy	61
6.4.5	State	62
7	Design Overview	63
7.1	Components that Interact	63
7.2	Design Challenges	65
7.2.1	Product Environment	65
7.2.2	Quality Requirements	66
7.2.3	Resulting Challenges for our Design	66
7.3	High-level Flow for Fixed Providers	67
7.4	Generic High-level Flow	68
7.5	Use Cases	70
8	Detailed Requirements	75

9	High-level Design	76
9.1	High-level Architecture	76
9.2	Internal Interface	77
9.2.1	Abstract Interface	77
9.2.2	DynAdiContainers	78
9.2.3	DynAdiEntitlementService as WebService	80
9.3	External Interface	82
9.3.1	General Considerations	82
9.3.2	Information Requirements	82
9.4	Static Class Design	83
9.4.1	The Class DynAdiEntitlementService	83
9.4.2	The WebService Adapter	83
9.4.3	The Classes Session and SessionTable	85
9.4.4	The Class DynAdiClient	85
9.4.5	The Class DynAdiProtocol	87
9.5	Usage of Design Patterns	87
9.5.1	Single Instance per ID	87
9.5.2	Design of Stateful Protocols	89
9.6	Dynamic Behavior and Collaboration	89
9.6.1	Rough Flow of a getEntitlements() Request	90
9.6.2	Example Protocol Flow	90
10	Implementation Issues	95
10.1	Testing	95
10.1.1	Exemplary Specification	95
10.1.2	The junit Test Suite	96
10.1.3	Simple Instance of Exemplary Specification	96
10.1.4	Advantages of Exemplary Specification	99
10.2	Optimization	99
10.2.1	Optimization According to Strict Rules	99
10.2.2	Performance Bottlenecks	100
10.2.3	Resource Pool Design Pattern	101
10.2.4	Usage of the jInsight Profiler	102
10.2.5	Performance Improvements	104
10.3	Metrics for the DynAdiEntitlementService	104
11	Conclusion and Future Work	107

A Detailed Requirements	108
A.1 Goals	108
A.1.1 Must-have Criteria	108
A.1.2 Want-have Criteria	108
A.1.3 Non-Goals	109
A.2 Product Use	109
A.3 Product Environment	109
A.3.1 Hardware	109
A.3.2 Software	109
A.3.3 Orgware	110
A.4 Product Function	110
A.4.1 Rough State Chart	114
A.4.2 Data Flow	114
A.5 Product Data	114
A.5.1 XML Format of the Container Descriptors	117
A.6 Product Performance	117
A.7 User Interface	118
A.8 Quality Requirements	118
A.9 Test Considerations	118
B Glossary	121
B.1 Terminology	121
B.2 Abbreviations	125

Chapter 1

Introduction

1.1 Problem Description

Protecting web-sites and applications against unauthorized access is one of the most important problems of the internet. This is even crucial for e-commerce, e-banking and e-government. The task to restrict access to these resources is fulfilled by so called Access Control Products that have two essential functions: On the one hand they provide authentication, which means that they identify a user. On the other hand they perform access control decisions that define whether a certain user is allowed to access a resource according to a given security policy.

Usually both functions are provided in different phases – an initial authentication phase and an access control phase afterwards. The user is normally authenticated only once. The access control decision takes place each time the user accesses a resource.

For both purposes Access Control Products require information. Within the authentication phase they need data to distinguish one user from another and to verify a user's identity. In this phase the Access Control Products issue a username and group memberships to the user. These are associated with specific access permissions. Within the access control decision they require a security policy in general and data that allow its evaluation for a specific user-resource combination.

Usually the Access Control Products take decisions according to different types of information:

- A security policy: It specifies which user is allowed to access a certain resource under which circumstances. It also defines which actions he is allowed to take.
- A user's credential: A credential contains data about the user such as username, group membership or extended attributes. Normally it is build up in the authentication phase
- Entitlements: These are attributes associated to the user by another party.
- Application context data. This data complements the other information and is mostly provided by back-end applications.

Most of today's Access Control Products use this information in a static manner. They usually specify the security policy as static access matrix which contains the users, the resources and the actions allowed (see also [SS94, Pfi00]). The Access Control Products

express this access matrix normally in so-called access control lists (ACL) or capability lists. The access control lists are assigned to a resource and contain the users that are allowed to access the resource. A capability list belongs to a certain user and contains the resources he is allowed to work with. We present an example for an ACL and a capability list that both allow a user “thomas” to access a file “diploma_thesis.tex”:

```
file diploma_thesis.tex: user=thomas permissions={read, write}
user thomas: file=diploma_thesis.tex permissions={read, write}
```

In each access control decision, the access control software checks the security policy, whether a user with given username and group memberships has the permission to access a resource or not. The access is granted, if the required static rights are present.

Unfortunately, this static approach is not sufficient for modern applications of e-commerce and e-banking. Some use cases are:

- Fine-grained access control based on detailed user attributes. Common examples of such attributes are a user’s age or credit line, whether he has a car license or insurance contract.
- Further examples come up in situations, where the user is associated with a changeable state. Especially services for customer retention require access control based on i.e. a certain amount of Web-Miles, Membership Rewards¹, Miles&More² or bahn.comfort³ points.
- Another example are pay-per-use services, where the user needs to have payed before getting access.
- If a company wants to deal with anonymous users or allow to trade with customers without generating a new account. As the importance of privacy concerns of users raises, these services get more important.
- Cross-domain cooperations of multiple companies. A simple example is the case that the user needs to be a gold customer of another company. Consider the case that a company wants to provide a special service with multiple service levels to members of another company. The service provider wants to distinguish different users, while the receiving company does not want to disclose its user database.

It is a new approach to complement the static access rights with Boolean rules over changeable attributes. In this case the access control software determines dynamically, whether the user’s or the context’s attributes fulfill the given rules at the very moment of the access control decision. Two simple examples of such rules are:

```
IF (user_age < 18) THEN deny_access
IF (has_drvs_license && has_car_insurance) THEN allow_access
```

Currently all major players on the access control market have included the capability to evaluate such Boolean rules as part of the access control decision. But the availability of such a rules evaluation only solves one part of the problem. The rules usually contain dynamic attributes like the user’s age or validity of driver license in the example above. These attributes sometimes have a very short validity time and have to be queried at the very moment of the rules evaluation.

¹A customer retention system of American Express (<http://www.americanexpress.com>)

²A customer retention system of the Lufthansa (<http://www.lufthansa.de>)

³A customer retention system of the Deutsche Bahn (<http://www.bahn.de>)

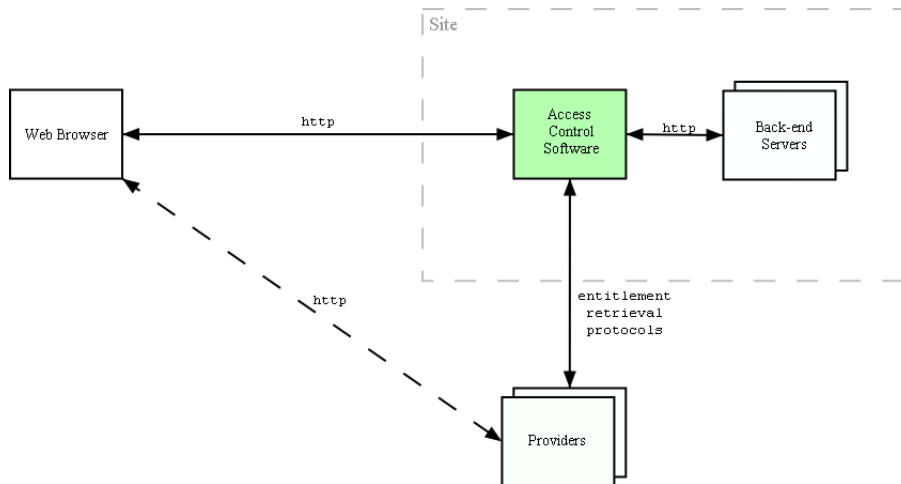


Figure 1.1: Situation of Access Control Products

This dynamical determination of the user's or context-dependent attributes (depicted in Figure 1.1) is a non-trivial task for an access control software. The variety of usable attribute types and diversity of the attribute provision market complicate this. Providers are for instance government or insurance agencies, banks or credit institutes as well as specialized entitlement providers like Verisign⁴ or Dun & Bradstreet (D&B)⁵. Each of them expects a different protocol to be used for the attribute query. And each of them delivers attributes in a different format. Given this problem it would be really helpful and a logical step to externalize this functionality to a service that does the attribute provision dynamically on demand.

1.2 Scope of this Thesis

In the theoretical Part I, we analyze some of the emerging protocol standards usable for entitlements provision. We focus on the security of OASIS Security Assertion Markup Language (SAML). It is used for the attribute provision of the entitlement service of our design part. Since the protocol included in our service is a simple challenge-response protocol, we extend our analysis to the Single Sign-on Browser/Artifact Profile of SAML. This profile is thought for the authentication phase of access control and provides dynamic assertions about the user's identity. There is no other work known to the author that has tried to analyze a standard like SAML in such an exact way.

We formalize real world concepts like a web browser and refine the protocol description to one suitable for a security analysis. We analyze the profile rigorously and try to discover security flaws and possible attacks on it. Though the protocol was thought secure, we found several attacks on the profile. We present a repaired version of the protocol that is secure against the described attacks.

The practical Part II of this thesis describes design and implementation of an entitlement service. This service queries the necessary attributes for access control decisions dynamically. On the one side it is able to connect to fixed providers and retrieve attributes from them. On the other side the service is capable of obtaining the browser-focus of a user's browser and running complex protocols with it.

⁴<http://www.verisign.com>

⁵<http://www.dnb.com/us>

We integrate this entitlement service in a given access control product requiring only minimal changes to the product itself. We include a protocol module for attribute retrieval from one certain provider. This protocol module uses the OASIS Security Assertion Markup Language (SAML) for the communication with the provider.

Our goal was to provide a product-quality, robust service that allows virtually all kinds of dynamical attribute retrieval technically possible. The service fulfilled the security and reliability requirements of professional Access Control Products and was able to cope with the through-put of company site access points. It is modular and easily extensible to anticipate future changes in this fast developing market.

As you can see in the portfolio diagram in Figure 1.2 we discuss context-based access control in two dimensions: On the one hand considering both phases (authentication and access control decision) and on the other hand with theoretical and practical aspects.

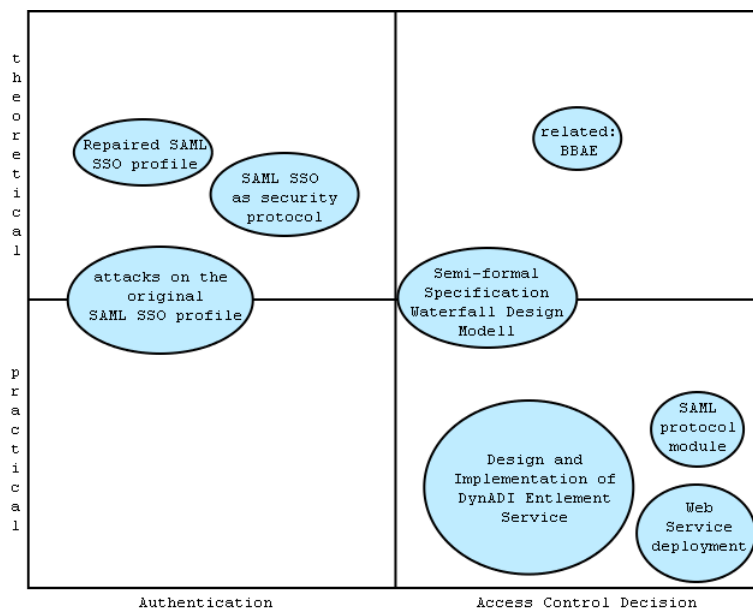


Figure 1.2: Portfolio Diagram of the Topics of this Diploma Thesis

1.3 Outline

This thesis is organized in two parts.

Part I: Protocol Analysis of the Security Assertion Markup Language In this part we analyze the security of the OASIS Security Assertion Markup Language (SAML). We begin with a general introduction of SAML and other state-of-the-art browser-based protocols in Chapter 2. There we explain the basic data structures and message formats of SAML, too.

Chapter 3 contains the analysis of the SAML Single Sign-on Browser/Artifact profile. We refine the given profile to protocol description suitable for a rigorous security analysis. We discover several attacks on the original profile. In Chapter 4 we present a repaired Single Sign-on profile that is resistant against these attacks.

We conclude the protocol analysis of Part I in Chapter 5.

Part II: Design of the DynAdiEntitlementService We describe the design and implementation aspects of the development of an attribute entitlement service. We introduce this topic in Chapter 6. We describe the market of access control products, the related literature to this part of the thesis and some of the design patterns we used.

In Chapter 7 we give a short overview about the design of our service. We describe the design challenges we face and analyze the feasibility of the project. There we present the most important use cases and a protocol flow solving the major problem.

Chapter 8 contains a detailed requirement description for our project. It includes goals and non-goals as well as quality requirements of our product.

We analyze the high-level design of the project in Chapter 9. We propose a design that solves the given challenges and describe the most important classes and solutions. This chapter contains the static class design as well as the dynamic behavior.

In Chapter 10 we consider chosen issues of the implementation. On the one hand this chapter contains our approach for the module testing. On the other hand we describe details about optimization measures and the metrics of the service.

We conclude Part II in Chapter 11. We describe the results of our design and items for future work in this chapter.

Part I

Protocol Analysis of the Security Assertion Markup Language

Chapter 2

Browser-based Protocols

In this part of the thesis we consider so-called browser-based protocols. Members of this protocol class have in common that they involve a web browser. Most of these protocols have the advantage over other authentication protocols like e.g. Kerberos [RFC1510], NIS+ or proprietary solutions that the browser is not protocol enabled. This means that the browser needs not to be enhanced with further software components. The protocols only uses the redirect and security capabilities any standard web browser has.

In this thesis we focus on the OASIS Security Assertion Markup Language (SAML), which is described in [SAML02a], and its Single Sign-on Browser/Artifact profile [SAML02b]. This protocol is an example of an emerging browser-based protocol standard. We analyze the SAML Single Sign-on rigorously and try to discover possible attacks on it. We do not know another work that tries such an exact analysis of a protocol standard like SAML SSO.

In this chapter we introduce browser-based protocols used for attribute and authentication queries. These protocols deal on the one hand with assertions about a user's identity and on the other hand with attributes that influence access control decisions. As stated in the following definitions, we call this kind of data Access Decision Information (ADI).

Definition 1 *We call the set of information taken into account to decide authorization queries Access Decision Information (ADI).*

In general these are credentials, entitlements, Protected Object Policy (POP) entries and information about HTTP requests.

Definition 2 *All Access Decision Information determined at the time the access decision is taken is called dynamic. We abbreviate this kind of information with the term DynADI.*

Dynamic Access Decision Information (DynADI) can be determined by using three kinds of protocols. We call the service that does the ADI retrieval entitlement service.

Fixed-provider The entitlement service calls an attribute provider directly using a given protocol that is suitable for that provider.

Browser-based The entitlement service contacts the user's browser and asks the user for the requested attributes or additional information needed to run other protocols.

Federated The entitlement service combines fixed-provider and browser-based approaches.

The characteristics of fixed-provider protocols are more relevant in the design part of this thesis. We describe them in Section 6.3 (Page 58). In this chapter we concentrate on state-of-the-art browser-based protocols. The most important ones for our project are the SAML Single Sign-on Browser/Artifact Profile (SAML SSO) and Browser-based Attribute Exchange protocol (BBAE). We describe SAML SSO in Section 2.1. This section also contains an introduction to the data-types and message formats of SAML. Section 2.2 contains BBAE and other important browser-based protocols such as Liberty, Shibboleth and Passport.

2.1 Introduction to the Security Assertion Markup Language (SAML)

The Security Assertion Markup Language (SAML) is a message standard of the OASIS consortium¹ that encodes security assertions and corresponding protocol messages in XML format. The message standard itself is described in [SAML02a]. This document defines the basic data-types, the so-called SAML assertion as well as the request and response message formats.

SAML allows so-called protocol bindings that embed SAML constructs in other structures for transport. Examples of such bindings are HTTP Form POSTs and XML-encoded SOAP messages. These bindings are described in [SAML02b].

Additionally the SAML standard includes descriptions of the use of SAML assertions in communication protocols and frameworks. These so-called profiles contain protocol flows and security constraints for applications of SAML. We consider the SAML Single Sign-on browser/artifact profile that is an example of such a protocol description. The profiles are also contained in [SAML02b].

The Security Assertion Markup Language (SAML) is one of the state-of-the-art message standards in the sector of security assertions. It is multi-functional and allows authentication, attribute and authorization decision assertions. It is the base for different fixed-provider and browser-based protocols.

We describe the data types and messages relevant for our security analysis of the SAML Single Sign-on Browser/Artifact Profile in Section 2.1.1. Section 2.1.2 contains an introduction to the SAML SSO profile. We show the details of the profile itself and the corresponding analysis in Chapter 3 (Page 20).

2.1.1 Example of SAML Data Types and Messages

In this section we introduce the data types and messages of the Security Assertion Markup Language (SAML). We omit XML elements that we do not need in the profile description and the security analysis. Please read the SAML specification [SAML02a] and the corresponding XML schemas [SAML02d] for further details.

In Figure 2.1 we present an example of a SAML Response, where we have omitted some not so important elements and the XML namespaces. Please note that this Response is not a correct SAML Response, it is meant to illustrate this message standard. In the following paragraph we explain the most important data-types of SAML. Most of them are used in the example of Figure 2.1, too.

¹<http://www.oasis-open.org>

```

<Response>
  <ResponseID>dynadi:2003-02-20:5aff53b33d7a78c4</ResponseID>
  <InResponseTo>access:2003-02-20:1e5638a2bbd3180a</InResponseTo>
  <IssueInstant>2003-02-20T09:12:15</IssueInstant>
  <Assertion>
    <AssertionID>dynadi:2003-02-20:777a35f6b3ac638b</AssertionID>
    <Issuer>urn:com:ibm:zurich:dynadi</Issuer>
    <IssueInstant>2003-02-20T09:11:07</IssueInstant>
    <SubjectStatement>
      <Subject>
        <NameIdentifier>john_smith@domain.org</NameIdentifier>
      </Subject>
    </SubjectStatement>
    <AuthenticationStatement>
      <AuthenticationMethod>urn:ietf:rfc:2246</AuthenticationMethod>
      <AuthenticationInstant>2003-02-20T09:10:59</AuthenticationInstant>
    </AuthenticationStatement>
  </Assertion>
</Response>

```

Figure 2.1: Example of a SAML Response (Some Elements Omitted)

IDType [SAML02a, p. 10] This basic data type declares identifiers for assertions, requests and responses. Values of this type are supposed to be chosen uniquely such that no party accidentally assigns the same identifier to a different data object. The `IDReferenceType` is used to refer to instances of `IDType`.

Assertion [SAML02a, p. 11] An Assertion is a package of information that supplies one or more statements made by an issuer. An Assertion contains the following major elements:

- *AssertionID* [`IDType`, required] Identifies the assertion.
- *Issuer* [`String`, required] The name of the issuer of the Assertion.
- *IssueInstant* [`UTC`, required] The time instant of issue.
- *Statement* [`Statement`, one or more] A Statement about a subject made by the issuer.

Statement [SAML02a, p. 15] The Statement element is an abstract extension point for different kinds of concrete Statements. The most important derived concrete elements are `SubjectStatement` and `AuthenticationStatement`. We present the hierarchy of the different statement types in Figure 2.2.

SubjectStatement [SAML02a, p. 15] This Statement contains a `Subject` element that allows an issuer to describe a subject. The `Subject` element itself has the following structure:

- *NameIdentifier* An identification of a subject by a name and security domain.
- *SubjectConfirmation* Information that allows the subject to be authenticated. If the `Subject` contains the `SubjectConfirmation` in addition to a `NameIdentifier`, the relying party can perform the `SubjectConfirmation` to verify that the entity presenting the assertion is the one that the issuer identifies with the `NameIdentifier`.

AuthenticationStatement [SAML02a, p. 18] The issuer states that the subject was authenticated by a particular means at a particular time.

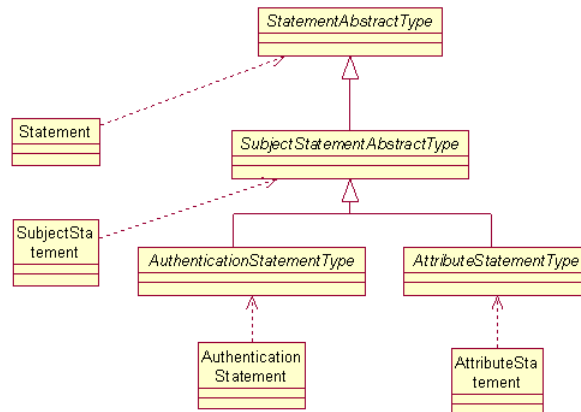


Figure 2.2: Hierarchy of the Statement Types

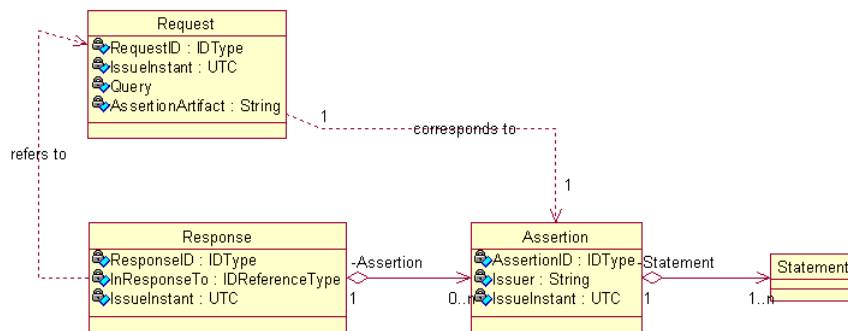


Figure 2.3: Relationship between SAML Request and Response

AttributeStatement [SAML02a, p. 21] The issuer states that the specified subject is associated with the specified attributes.

Request [SAML02a, p. 24] The requester sends a message of type Request to a SAML responder. The Request inherits different elements from RequestAbstractType. We depict the relationship between Request and Response in Figure 2.3.

- *RequestID* [IDType, required] Identifies the request. If present the Response corresponding to the Request must contain its value in the InResponseTo element.
- *IssueInstant* [UTC, required] The time instant of issue of the request.
- *Query* Specifies the query to be answered by the SAML responder.
- *AssertionArtifact* [String, one or more] Contains a so-called SAML artifact, which is a String handle and represents a specific assertion.

Response [SAML02a, p. 29] The Response element specifies the message sent back by the SAML responder.

- *ResponseID* [IDType, required] Identifies the response.
- *InResponseTo* [IDReferenceType, optional] Refers to the Request the Response corresponds to. If the RequestID of the Request can be determined, it must be present.

- *IssueInstant* [UTC, required] The time instant of issue of the Response.
- *Assertion* [Assertion, any number] Zero or more Assertions about a subject.

2.1.2 The SAML Single Sign-on Profile (SAML SSO)

The SAML Single Sign-on Browser/Artifact Profile (abbreviated with SAML SSO) is one profile for SAML and specified in [SAML02b]. It describes the usage of SAML messages to perform a single sign-on operation involving three parties – a user equipped with a standard browser, a source site, and a destination site. We depict the protocol flow of SAML Single Sign-on in Figure 2.4.

The main advantage of the SAML SSO over different other single sign-on protocols is that the user only needs a standard web browser. This browser is not SAML enabled, which means that it is not extended by a special protocol module and does not need to support active content or scripting. The user can browse with any browser of his choice and has not to rely on specialized clients.

The user logs in at a source site typically after a redirect from a destination site. He authenticates to that site. The source site stores an assertion about the user's identity and redirects the user's browser to the destination site the user wants to browse.

The source site includes a small piece of data called SAML artifact into the redirect, which refers to the stored assertion. Receiving the redirect with this artifact, the destination site shows this artifact to the source site and requests the corresponding assertion from it. By providing this assertion to the destination site, the source site confirms that the user presenting the SAML artifact was authenticated by the source site.

We describe the details of the SAML Single Sign-on Browser/Artifact profile in Chapter 3 (Page 20).

2.2 Related Protocols and Projects

Within the area of federated identity management and therefore context-based access control and attribute exchange we find several other major projects and protocol classes. We do not use them within this thesis, but want to survey the field we work in.

2.2.1 The Browser-based Attribute Exchange Protocol (BBAE)

The Browser-based Attribute Exchange (BBAE) is a new protocol for browser-based attribute retrieval invented by Birgit Pfitzmann. It is sketched in [PW02b] and described in more detail in [PW02a]. It allows authentication as well as attribute exchange in a federated identity environment. The BBAE protocol could be implemented by the utilization of SAML Assertions to transfer attributes. In such an implementation it would be closely related to the SAML Single Sign-on Browser/Artifact profile (Section 2.1.2). In contrary to SAML SSO the Browser-based Attribute Exchange is directly applicable to the attribute retrieval of the `DynAdiEntitlementService` that we design in Part II. We depict a simplified protocol flow of the BBAE protocol in Figure 2.5.

The BBAE protocol is specialized in attribute provision whereas the SAML SSO profile concentrates on authentication. Additionally the BBAE is designed to be privacy friendly which means it allows anonymity on the higher protocol layers even for a local wallet (i.e. for users holding their attributes and authentication locally).

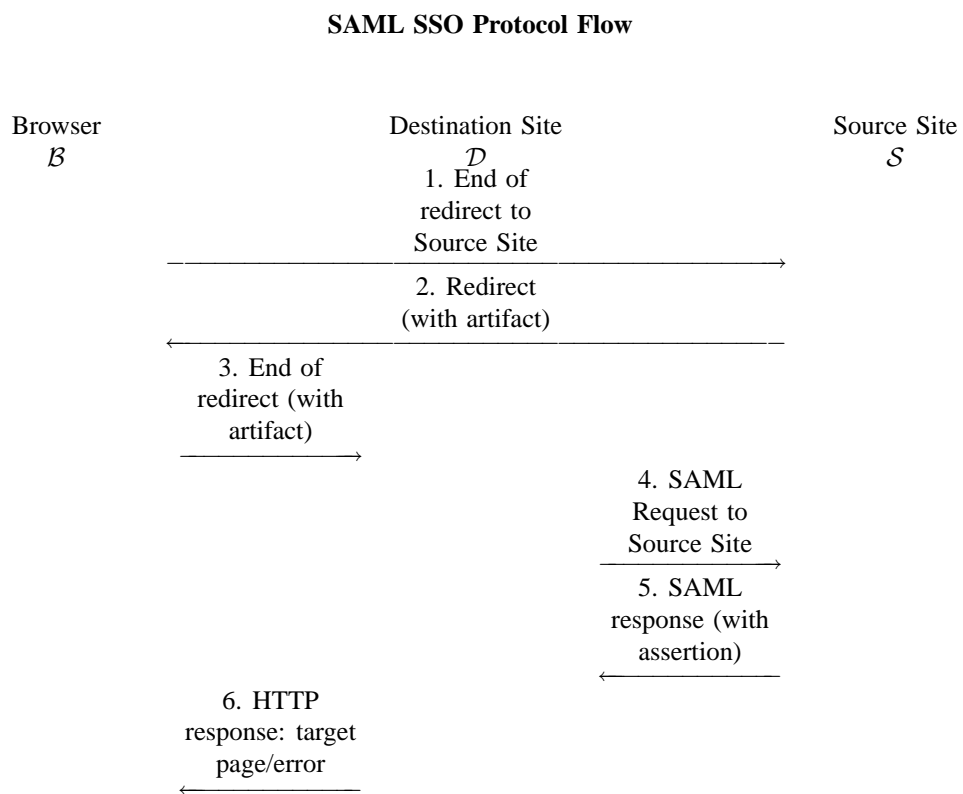


Figure 2.4: Protocol Flow of the SAML Single Sign-on Browser/Artifact Profile

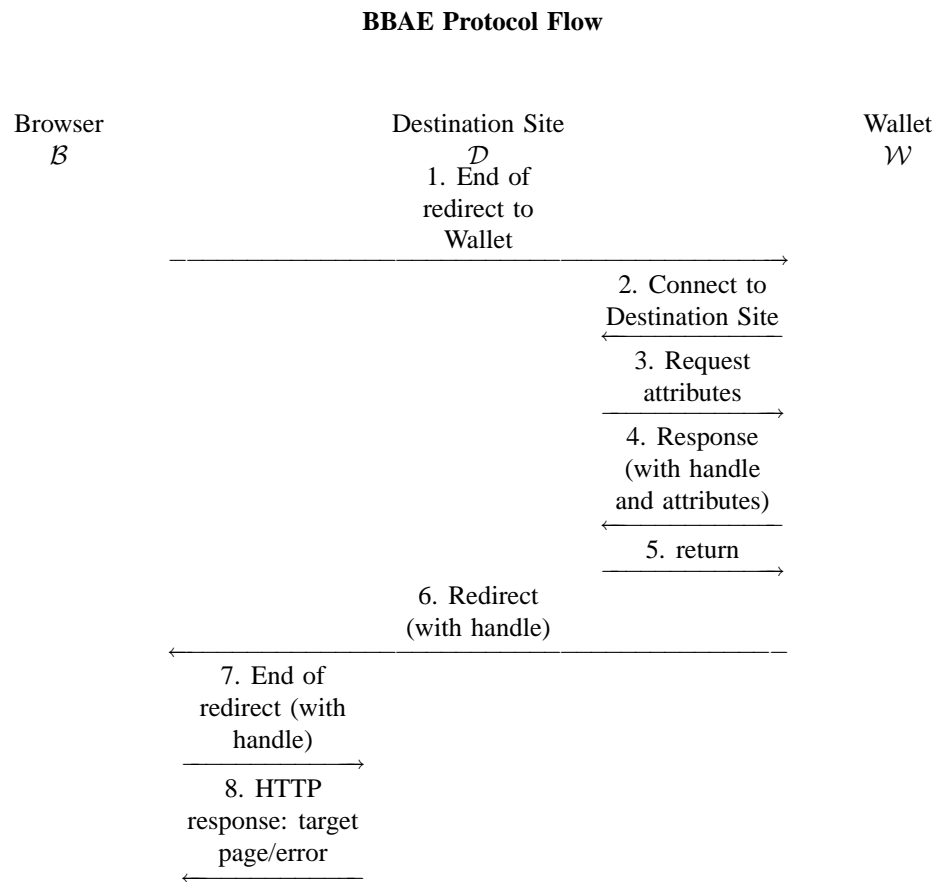


Figure 2.5: Protocol Flow of the Browser Based Attribute Exchange (BBAE)

2.2.2 Liberty

The Liberty Alliance Project² has the goal to establish an open standard for federated network identity. The Liberty architecture contains new message formats extending the ones of SAML and four concrete profiles. The architecture of Liberty is described in several documents, where the most important are: the architecture overview [LIB03a], the bindings and profiles specification [LIB03b] and the protocols and schema specification [LIB03c].

As far as we can see these protocols are likely to become important in practice. Three of these protocols assume a thin client such as a web browser not aware of the protocol. The fourth one, the Liberty-enabled Client and Proxy Profile (LEC) assumes a fat client, a special client capable of this protocol.

The original LEC protocol was affected by a man-in-the-middle attack that is described in [PW02c]. As the Liberty Architecture enhances the SAML message formats and therefore inherits many properties of SAML, it might also be vulnerable to some of the flaws we describe in Chapter 3 (Page 3). But an analysis of the Liberty protocols is out of the scope of this thesis.

2.2.3 Shibboleth

Shibboleth³ is a joint project of Internet2/MACE and IBM developing architectures and policy structures. It provides support of inter-institutional sharing of web-resources. The Shibboleth Project especially provides access control based on attributes, which is quite related to our own work on context-based access control. The Shibboleth Architecture is described in the current draft [SHIB02].

Shibboleth uses the Security Assertion Markup Language (SAML) for the message and assertion formats. The protocols bindings based on SAML as well.

2.2.4 MS Passport

Microsoft Passport⁴ [Mic02a] is a suite of Web-based services including single sign-on. Passport is one of the first wide-spread single sign-on solutions. As far as we know Passport is integrated in the Microsoft products MSN, Hotmail and Windows XP, which supports such a strong market penetration. You can find a more technical specification of the Passport service in the Passport Review Guide [Mic02b].

Marc Slemko [Sle01] as well as David P. Kormann and Aviel D. Rubin [KR00] revealed several attacks on Microsoft Passport that are similar to some of the attacks we present in this thesis in Section 3.4 (Page 40).

²<http://www.projectliberty.org>

³<http://shibboleth.internet2.edu>

⁴<http://www.microsoft.com/netservices/passport/overview.asp>

Chapter 3

Security Analysis and Attack of the SAML Single Sign-on Profile

3.1 Introduction

The SAML Single Sign-on browser/artifact profile (SAML SSO) as described in Section 2.1.2 is a typical example of a browser-based protocol based on SAML. We introduced the Security Assertion Markup Language (SAML) and this profile in chapter 2.

The advantage of SAML SSO to utilize a non-enabled browser implies the main challenges of the security design of such a browser-based protocol. The designers have to cope with the restricted capabilities of a standard web browser:

HTTP as communication protocol: HTTP [RFC2616](respective HTTP over SSL/TLS [RFC2818]) is the only protocol commonly available to transfer data via the browser. The only methods to send data to another party are HTTP Form POSTs and the URL of a GET request.

Restricted message length: If data are transferred in the request URL, the length of this URL is limited to a small implementation dependent value. On the one hand the length that different browser types support differ, on the other the implementation of routers and other network hardware might be different. Therefore it is not possible to send arbitrary messages.¹

Statelessness: The browser does not keep a state for the protocol flow of the single sign-on protocol.

Weak authentication: The browser has not got a client certificate and cannot participate in bilateral authentication. The authentication of servers strongly depends on the security of the implemented Public-Key Infrastructure.

The SAML SSO browser/artifact profile uses different techniques to cope with these problems. It uses HTTP redirects to send information to other parties. Within the URL of these redirects it includes small pieces of data called artifacts that refer to larger SAML assertions transferred through other channels.

¹HTTP/1.1 [RFC2616] does not state a restriction on the URL length, but recommends not to rely on a URL length larger than 255 bytes.

In this chapter we describe the details of the protocol flow and analyze its security. In Section 3.2 we consider the general problems we faced during the analysis of the profile and the methods we applied. We describe the participating parties, the interface of the protocol and the trust model there. This section also contains the protocol schema as defined by the original SAML profile. In Section 3.4 we show different attacks on the given schema.

We improve the protocol schema in Chapter 4 (Page 47).

3.2 Refinement of the SAML SSO Profile

It is a non-trivial task to analyze the security of a protocol standard such as the Security Assertion Markup Language (SAML) and its Single Sign-on Browser/Artifact profile. For such an analysis we need precise statements about the protocol and message formats. Because of this need, we faced the following general problems during the analysis:

Imprecise English language: Descriptions written in English language can bear ambiguities. They may reference cryptographic keywords without specifying the necessary context.

No formal methods The protocol description often lack formal descriptions of the protocol flow such as state charts or protocol interfaces.

No robustness measures: Some protocol standards such as the SAML SSO Profile lack explicit robustness measures, e. g. replay prevention, actuality, or explicit naming of a message (protocol name, protocol step and unique transaction id).

Distribution of relevant information: The documents are often organized according to design-related chapters and the internal structure of the protocol. This structure distributes the relevant information for a security analysis over different documents.

Implicit assumptions: Assumptions about the behavior of the participating parties and cryptographic primitives are hidden.

Given these problems, we decided to start the analysis with a refinement of the concrete SAML profile. It is a common approach in cryptography to invent an abstract model for a protocol and analyze the security properties of this model. For instance security analyses often assume secure channel or random oracle models as a base. Unfortunately, this approach has the big disadvantage that the security properties derived from the abstract model do not necessarily apply to the real world.

Therefore we chose a technique of refinement to get a more formal description of the protocol that allows a rigorous security analysis and is as similar as possible to the original profile. We want to extract the essential information from the profile without changing its meaning. We do this by rewriting the protocol description in an established structure. We quote the original profile as often as possible and name the requirements and prohibitions given in the profile explicitly as so-called protocol constraints.

Our refinement also involves a more precise formulation of certain statements. Since the original statements allow a wider range of interpretations, we only analyze the security of a subclass of the original SAML Single Sign-on Browser/Artifact profile. Yet, our sub-class covers most of the cases of the original protocol.

To refine the profile we propose the following technique:

Read the standard: When reading the standard it is important to keep in mind which sentences are normative and which are not. Usually the key words “MUST”, “MUST NOT”, “REQUIRED”, “SHALL”, “SHALL NOT”, “SHOULD”, “SHOULD NOT”, “RECOMMENDED”, “MAY”, and “OPTIONAL” are used in sentences that are normative. Their correct usage is described in [RFC2119]. In SAML [SAML02a] these keywords are used to specify requirements over protocol and application features and behavior that affect the interoperability and security of implementations.

Please note: We only consider criteria of absolute requirement or prohibition (MUST, SHALL, REQUIRED, MUST NOT, SHALL NOT) for the security analysis. In other cases an implementation could fulfill the standard without fulfilling the given constraint and perhaps break the protocol security.

Extract relevant constraints: We suggest the following steps to extract all relevant protocol statements from the standard. We do this to bring together all security relevant information and to rewrite the protocol in a compact structured manner.

We extract each explicit required or prohibited criteria that

1. is given as constraint of the protocol flow.
2. is given about data structures and message formats referenced in the protocol description.
3. is given in chapters that are related to the type of protocol, its application, participating parties, etc.

Please note that we omit the documentation of this extraction process within this thesis. We only provide the rewritten protocol schema.

Structured protocol rewrite: The next step is the rewriting of the protocol in a structured format suitable for cryptographic protocols. We recommend the following structure, which is similar to the one proposed in [Pfi99].

Participating Parties: A description of the participants of the protocol. It includes corresponding keys, certified identities and addresses that exist before the protocol run.

Protocol interface: The protocol interface contains a short description of the protocol without implementation details.

- *Parameters:* The general and security parameters of the protocol.
- *Roles:* The different roles a participant can act in.
- *Subprotocols or services:* The services that the protocol provides including the input and output of the different roles in these services.
- *Requirements:* Relevant security requirements for the protocol.

Trust model: The trust model specifies general security assumptions about the communication channels and the adversary the security analysis deals with. It specifies the context of the analysis explicitly.

Protocol schema: The protocol schema is a detailed description of the protocol flow. We consider the schema as implementation of the protocol interface and thus describe it in an own section. The schema states the behavior of the participants explicitly. It contains the protocol constraints applying to the different steps, data structures and messages.

Depending on the concrete protocol it is helpful to depict the protocol flow in a sequence diagram, a state chart or a collaboration diagram.

Extend the protocol skeleton We invent missing parts of the skeleton on our own. We make up explicit assumptions for all implicit or unclear assumptions in the standard. If not given in the protocol standard, we invent an explicit trust model and adversary description.

Using the technique described above we generate two major types of statements about the protocol: constraints and assumptions. We define them as follows:

Definition 3 (Constraint) *A Constraint is a statement given in the original SAML Single Sign-on Browser/Artifact Profile that specifies requirements over protocol features and behavior that affect the interoperability and security of implementations.*

Definition 4 (Assumption) *A Assumption is a statement we invent in extension of the protocol skeleton in order to formulate precise security requirements.*

We classify constraints and assumptions according to the protocol phases they are applicable to. The general ones hold for several parties and protocol phases. While protocol assumptions describe the protocol flow of a certain party, setup assumptions refer to a party's setup phase.

3.2.1 Notation

We use the following terms:

- \mathcal{P} is a term for a party. Please note that this identifier is an abstract reference to the party in the analysis and not the name of \mathcal{P} certified to another party.
- In situations where there are two parties with the same role in the protocol and one impersonates the other, we distinguish \mathcal{P} and \mathcal{P}' . In this case \mathcal{P} is the honest party that is supposed to be in that role. \mathcal{P}' is another party that impersonates \mathcal{P} in that role.
- If a party \mathcal{P} impersonates possibly several other parties intentionally, we also use another notation: Then \mathcal{P} is written with reference to the impersonated party \mathcal{X} as index ($\mathcal{P}_{\mathcal{X}}$).
- The String variable $id_{\mathcal{P}}$ refers to an identity of party \mathcal{P} . For instance, this can be an identity in a certificate or a NameIdentifier of a SAML assertion.
- We distinguish messages and pieces of data. A message m is a string sent by a party. A piece of data $\langle \text{DATA} \rangle$ (for instance a URL) can be part of such a message, but is not sent independently.
- A term $m^{(\mathcal{P})}$ denotes the message m in the view of \mathcal{P} . We use a similar notation for data: $\langle \text{IST-URL} \rangle_{\mathcal{Q}}^{(\mathcal{P})}$ denotes the $\langle \text{IST-URL} \rangle$ of \mathcal{Q} in the view of \mathcal{P} .
- We use symbolic references like [ASB7] to refer to assumptions and constraints. The first letter refers to the general type (A: assumption; C: constraint), the second letter to the classification (G: General; P: Protocol; S: Setup). The third letter refers to the participating party the assumption or constraint holds for (B: browser; D: destination site; S: source site; U: user). So the example above denotes a setup assumption about a browser.

3.2.2 Participating Parties

In this section we describe the parties that participate in the protocol. A sketch of the protocol is shown in Figure 2.4. In Figure 3.3 on Page 36 we depict the collaboration between the different parties.

The SAML Single Sign-on defines different attributes assigned to each party within a setup phase. We describe this data in Section 3.2.5 of Page 28. We describe the assumptions about the different parties in Section 3.3 (Page 40).

User U who wants to access a destination site using a web browser.

Attributes Corresponding to User U :

1. String id_U : The user U has a certain single sign-on identity id_U that is known to the source sites U registered at.
2. String $login_U$: For each source site the user U registered at holds: The user U knows some single sign-on information $login_U$ that the source site associates with id_U .

Browser B the user U accesses the internet with during the SAML SSO protocol flow. This is a standard browser in a secure configuration. The browser B is not SAML enabled. The browser B trusts a set of certification authorities.

Source site S is the site of a single sign-on provider, where the user U performs the single sign-on. The source site S knows the user U 's identity id_U and single sign-on login information $login_U$. Source site S and each destination site it communicates with know each other's identities. The source site S knows a set of trusted certification authorities

Attributes Corresponding to Source Site S : Each source site S chooses the following attributes. For each destination site the source site S cooperates with these attributes must refer non-ambiguously to S .

1. Certificate $cert_S$: A certificate used in channels that provide authentication. It should be signed by a certification authority that all participants trust.
2. String id_S : The identity in the certificate $cert_S$ of S .

Destination site D the user U wants to browse. The destination site D trusts a set of certification authorities.

Attributes Corresponding to Destination Site D : The destination site D chooses the following attributes. For each source site S the destination site D trusts these attributes must refer non-ambiguously to the party D :

1. Certificate $cert_D$: A certificate used in channels that provide authentication. It should be signed by a certification authority that all participants trust.
2. String id_D : The identity in the certificate $cert_D$ of D .

Please Note: There can be many instances of the different roles. In the following analysis we consider the case that a user U has exactly one source site S . The user U wants to access one destination site D .

3.2.3 Protocol Interface

Parameters

The SAML Single Sign-on Browser/Artifact profile does not name parameters explicitly. To analyze the security of the profile with a polynomial adversary we need to use parameters that specify the strength of certain cryptographically relevant measures:

Security parameter l This value specifies the security of cryptographic measures like encryption or signing.

Length of identifiers k The length of message and assertion identifiers has an influence on the probability that an adversary can produce collisions with valid but secret identifiers.

Length of the assertion handle h The SAML artifacts contain a handle to an assertion, that is kept secret. The length of this nonce strongly influences the probability that an adversary can guess or invent an artifact referring to an outstanding assertion.

Please note: For a given polynomially bounded adversary \mathcal{A} these parameters have to be chosen corresponding to the security assumptions we make. Especially if the profile or we classify a success probability of an adversary as negligible, these values of parameters have to be variable. The parameters k and h can be dependent from l with suitably chosen functions $k = f_k(l)$ and $h = f_h(l)$.

Sub-protocols

The protocol has one main service called *single_sign_on*. The protocol requires a separate setup phase that we do not specify as sub-protocol. The *single_sign_on* can only be called after the source and destination sites that participate in the protocol run have finished the *setup* phase successfully. To provide his single sign-on login information $login_{\mathcal{U}}$ to a source site \mathcal{S} both parties perform the sub-protocol *sso_login*.

We describe the inputs and outputs of the sub-protocols using the following interface:

$$I_{\mathcal{S}} \times I_{\mathcal{D}} \times I_{\mathcal{B}} \times I_{\mathcal{U}} \longrightarrow O_{\mathcal{S}} \times O_{\mathcal{D}} \times O_{\mathcal{B}} \times O_{\mathcal{U}}$$

In this interface $I_{\mathcal{P}}$ is the input and $O_{\mathcal{P}}$ the output set of a party \mathcal{P} .

setup The SAML Single Sign-on profile requires a setup phase. Since this phase is not specified as protocol, we do not fix the interface of a setup subprotocol either.

sso_login A user \mathcal{U} logs in at a source site \mathcal{S} with his single-sign on login information $login_{\mathcal{U}}$. The source site \mathcal{S} verifies \mathcal{U} login information and stores data for a later user tracking.

$$\begin{aligned} \text{single_sign_on} : & \quad (\cdot, \cdot, (login_{\mathcal{U}}), \cdot) \\ & \longmapsto ((id_{\mathcal{U}}/error), \cdot, (success/error), \cdot) \end{aligned}$$

The token $login_{\mathcal{U}}$ denotes the login information the user \mathcal{U} uses to perform the single sign-on at source site \mathcal{S} .

The source site \mathcal{S} outputs the user's $id_{\mathcal{U}}$ or an error message and sends a success or error page to the user \mathcal{U} 's browser \mathcal{B} .

single_sign_on He initiates the single sign-on. The source site redirects the user \mathcal{U} 's browser to the destination site \mathcal{D} , which then queries the source site \mathcal{S} for assertions about the user \mathcal{U} .

$$\begin{aligned} \text{single_sign_on} : & \quad (\cdot, \cdot, (\langle \text{IST-URL} \rangle_{\mathcal{S}}, \langle \text{Target} \rangle, \cdot) \\ & \longmapsto (\cdot, (id_{\mathcal{U}}/error), (page/error), \cdot) \end{aligned}$$

The URL $\langle \text{IST-URL} \rangle_{\mathcal{S}}$ is the inter-site transfer URL of source site \mathcal{S} . The piece of data $\langle \text{Target} \rangle$ describes the target the user \mathcal{U} wants to access.

The SAML SSO protocol does not state which party provides the inputs. The user \mathcal{U} can enter them into the browser \mathcal{B} as well as these inputs can be provided by the destination site \mathcal{D} or a third party.

The destination site \mathcal{D} returns either the identity of user \mathcal{U} $id_{\mathcal{U}}$ or an error message. In the case of the error message the authentication failed.

Requirements

Three main security requirements are:

Correctness Let every participant be honest and follow the protocol flow without interference of an adversary. If a user \mathcal{U} with name $id_{\mathcal{U}}$ performs the single sign-on login at his source site \mathcal{S} , he will be redirected to the destination site \mathcal{D} . The destination site \mathcal{D} will output the identity $id_{\mathcal{U}}$ or an error message.

Confidentiality The honest user \mathcal{U} 's login information are only known to user \mathcal{U} , \mathcal{U} 's source site \mathcal{S} and \mathcal{U} 's browser \mathcal{B} .

Authentication Let \mathcal{U} be an honest user. Let \mathcal{D} be an honest destination site. Let \mathcal{S} be one of the source sites the destination site \mathcal{D} trusts. Let source site \mathcal{S} refer to \mathcal{U} with identity $id_{\mathcal{U}}$. If the destination site \mathcal{D} believes it has a secure channel C to a person with name $id_{\mathcal{U}}$ according to a SAML SSO assertion of source site \mathcal{S} , this connection is indeed with user \mathcal{U} with the single sign-on identity $id_{\mathcal{U}}$ certified by \mathcal{S} .

3.2.4 Trust Model

Within this section we specify the general trust model we use for the security analysis.

In general we consider the participants of the protocol as possibly trustworthy and honest parties. To be more precise, the set of honest parties \mathbb{H} for one protocol run is:

$$\begin{aligned} \mathbb{H} &= \{\mathcal{U}, \mathcal{B}, \mathcal{D}\} \\ &\cup \{\mathcal{S} \mid \mathcal{S} \text{ is the source site of } \mathcal{U}\} \\ &\cup \mathbb{S} = \{\mathcal{S}_i \mid \mathcal{D} \text{ trusts } \mathcal{S}_i\} \end{aligned}$$

The source site \mathcal{S} is element of the set \mathbb{S} of source sites trusted by the destination site \mathcal{D} . The browser \mathcal{B} of user \mathcal{U} is considered to work correct according to our security assumptions within lifetime of the sessions that are involved in a protocol run.

Communication Channels

As depicted in Figure 3.1 the participants \mathcal{D} , \mathcal{S} and \mathcal{B} are connected to each other by channels in both directions. The user \mathcal{U} is connected to the browser \mathcal{B}

We assume an asynchronous channel model, where in general everyone is able to tap or interrupt channels as well as manipulate data transferred through them.

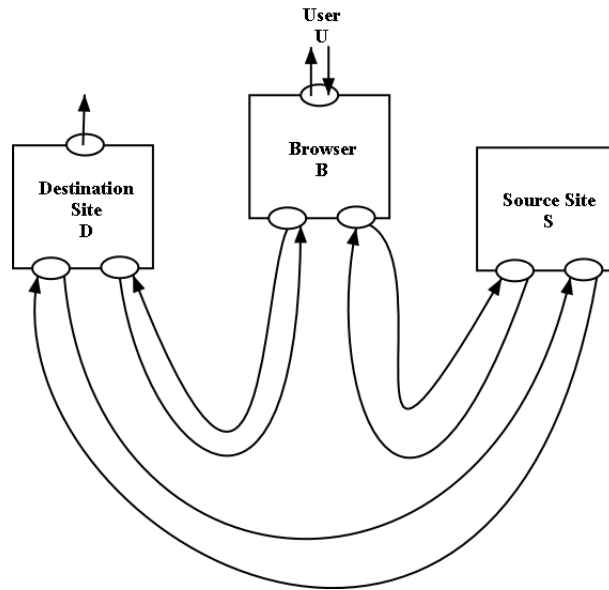


Figure 3.1: Channels Between the Participants

We assume the channel between browser B and user U as secure. A priori we consider the network channels as insecure. But we expect the connection security constraints of the profile to be fulfilled. Therefore the communication channels should have the following properties:

- Confidentiality of the messages in each protocol step as defined in Assumption [AP2] on page 40.
- Integrity of the messages in each protocol step as described in [AP1] on page 40.
- We only expect bilateral authentication for the SAML binding.
- For the improved SAML SSO Browser/Artifact profile we assume unilateral authentication of the server in all steps.
- We do not expect availability for the communication channels, but do not prove a availability property for the protocol either.

Adversary

We assume an adversary \mathcal{A} is element of the polynomial bounded Probabilistic Turing Machines (PPT). The trust model and the properties of the communication channels imply the following characteristics:

- The adversary \mathcal{A} can tap and interrupt all communication channels used by the protocol. He is able to manipulate data transferred through the communication channels.
- \mathcal{A} cannot corrupt the destination site D and all source sites S trusts. He cannot corrupt the browser B while the user U uses it during the protocol run. The adversary \mathcal{A} is able to corrupt the user U 's browser B before and after the session of the protocol run. In the first case \mathcal{A} cannot manipulate the browser B 's behavior such that it does not fulfill our security assumptions within this session.

- The adversary cannot influence the information transferred out of band during a protocol setup.

3.2.5 Protocol Schema

We now describe the protocol schema for the SAML Single Sign-on profile as described in [SAML02b]. We depict the protocol flow of the *single_sign_onservice* itself in Figure 3.2. You find a description of the data structures and messages of SAML in Section 2.1 on Page 13.

We collect all absolute requirements and prohibitions of the SAML Single Sign-on profile and divide them into to groups. We describe criteria independent of protocol steps as general constraints in the following section. We state that ones directly related to protocol steps as protocol constraints in the schema itself. We describe the additional assumptions we made in Section 3.3 (Page 40).

General Constraints

In this section we formulate the general constraints of the profile. We use the terms of the SAML core document [SAML02a] and the SAML bindings [SAML02b] to keep their exact sense. If we change the original text, we surround these changes with squared brackets. We use the following definition for the SAML searchpart the profile refers to:

Definition 5 (SAML searchpart) *The SAML binding [SAML02b] defines a searchpart of a URL as SAML searchpart, if it contains exactly one target description and one or more SAML artifacts. It has the following format:*

SAML searchpart: ... TARGET=<Target> ...
SAMLart=<SAMLart>₀ ... SAMLart=<SAMLart>_{n-1} ...

- where “TARGET” and “SAMLart” are String constants,
- $n \in \mathbb{N}$ is the number of artifacts,
- <Target> is a String description of the target resource and
- each <SAMLart>_i is a SAML artifact.

The general constraints for the SAML Single Sign-on Browser/Artifact Profile are:

1. [CG1]: The type IDType satisfies the following properties.
 - (a) Any party that assigns an identifier must ensure that there is a negligible probability that that party or any other party will accidentally assign the same identifier to a different data object. [SAML02a, p. 10]
 - (b) Where a data object declares that it has a particular identifier, there must be exactly one such declaration. [SAML02a, p. 10]

Note: This constraint is not strong enough to use elements of type IDType (for instance the RequestID) as a nonce: These constraints do not prevent an adversary from predicting an element of IDType easily or generating an instance of IDType that collides with an existing one intentionally.

2. [CG2]: For the RequestID hold the following statements. Please note that the RequestID is a subtype of IDType and that [CG1] holds for them, too.

- (a) The values of the RequestID attribute in a request [q] and the InResponseTo attribute in the corresponding response [r] must match. [SAML02a, p. 24]
 - (b) If the RequestID of a request [q] cannot be determined, then the InResponseTo of the response [r] must not be present. Otherwise it must be present and match the value of the corresponding RequestID attribute. [SAML02a, p. 30]
3. **[CG3]**: For the structure of a SAML searchpart as defined in Definition 5 hold these properties:
- (a) A single target description must be included in the SAML searchpart component. [SAML02b, p. 15]
 - (b) At least one SAML artifact must be included. [SAML02b, p. 15]
Note: The browser/artifact profile states that a user successfully registered at the source site. The profile does not explicitly say that an honest source site only proceeds with Step 2, if the user is authenticated. We follow this constraint and declare this as precondition of the protocol in Section 3.2.5.
 - (c) If more than one artifact is carried within SAML searchpart, all the artifacts must have the same SourceID. [SAML02b, p. 15]
4. **[CG4]**: The profile states following properties about the different parts of a SAML artifact.
- (a) Any two source sites with a common destination site must use distinct SourceID values. [SAML02b, p. 17]
 - (b) It must be infeasible to construct or guess the value of a valid, outstanding assertion handle. [SAML02b, p. 17]

Protocol Setup

The SAML single sign-on profile does not fix a setup procedure intentionally. Therefore we do not fix the steps that have to be done in this service, either. Still, the setup assumptions **[ASS1]** and **[ASD1]** given in Section 3.3.1 (Page 40) have to be fulfilled after a successful setup.

Therefore, in the setup phase the participating source sites \mathcal{S} and destination sites \mathcal{D} have to choose different pieces of data:

Source site \mathcal{S} : Each source site \mathcal{S} chooses the following attributes. For each destination site \mathcal{D} the source site \mathcal{S} cooperates with these attributes must refer non-ambiguously to \mathcal{S} .

1. 20-byte sequence $SourceID_{\mathcal{S}}$: The source site \mathcal{S} possesses a $SourceID_{\mathcal{S}}$, which refers to its $id_{\mathcal{S}}$.
2. URL $\langle IST-URL \rangle_{\mathcal{S}}$: The single sign-on profile names the URL where the user \mathcal{U} logs in and performs the single sign-on inter-site transfer URL $\langle IST-URL \rangle_{\mathcal{S}}$.
3. URL $\langle SR-URL \rangle_{\mathcal{S}}$: The source site provides a SAML responder URL $\langle SR-URL \rangle_{\mathcal{S}}$, where destination sites send SAML requests to and which sends the responses back.

Destination site \mathcal{D} : The destination site \mathcal{D} chooses the following attributes. For each source site \mathcal{S} the destination site \mathcal{D} trusts these attributes must refer non-ambiguously to the party \mathcal{D} :

1. URL $\langle \text{AR-URL} \rangle_{\mathcal{D}}$: The destination site \mathcal{D} provides an artifact receiver URL called $\langle \text{AR-URL} \rangle_{\mathcal{D}}$, where a source site \mathcal{S} can redirect a browser to for handing over artifacts.
2. URL $\langle \text{SQ-URL} \rangle_{\mathcal{D}}$: The destination site uses this URL as source for its SAML requests and as destination for other parties' SAML responses. If both messages are transferred within the same channel, $\langle \text{SQ-URL} \rangle_{\mathcal{D}}$ is the hostname of \mathcal{D} .

For each communication relationship between a source site \mathcal{S} and a destination site \mathcal{D} , both have to transfer these values out of band to the communication partner. Both partners have to verify that the received values are non-ambiguous in their context.

Each source site \mathcal{S} builds up a table of artifact receiver URLs. It contains one entry for each destination site \mathcal{D} the source site \mathcal{S} communicates with and links the $\langle \text{AR-URL} \rangle_{\mathcal{D}}$ to the destination site \mathcal{D} 's identity $id_{\mathcal{D}}$. The table is specified in Assumption [ASS1] on Page 40.

Each destination site \mathcal{D} generates a table of *SourceIDs*. The table contains one entry for each source site \mathcal{S} the destination site \mathcal{D} trusts. It links the $SourceID_{\mathcal{S}}$ and the SAML responder URL $\langle \text{SR-URL} \rangle_{\mathcal{S}}$ to the source site \mathcal{S} 's identity $id_{\mathcal{S}}$. The SourceID table is described in Assumption [ASD1] (Page 40).

Subprotocol: `sso_login`

We do not define the details of this subprotocol. There are common implementations for such a login procedure.

Subprotocol: `single_sign_on`

The SAML Single Sign-on Browser/Artifact Profile starts with a redirect s to the $\langle \text{IST-URL} \rangle_{\mathcal{S}}$ of a source site \mathcal{S} . This redirect is initiated by another party which not specified in the profile. This party provides the $\langle \text{IST-URL} \rangle_{\mathcal{S}}$ and a String target description $\langle \text{Target} \rangle$. The target description refers to a resource on a destination site \mathcal{D} that the user \mathcal{U} wants to browse.

Precondition: The user \mathcal{U} and a source site \mathcal{S} have run the *sso_login* sub-protocol: The user \mathcal{U} logged in at source site \mathcal{S} successfully. \mathcal{U} provided the login information $login_{\mathcal{U}}$ via \mathcal{U} 's browser \mathcal{B} . The source site \mathcal{S} verified this login information and returned $id_{\mathcal{U}}$. The validity time of \mathcal{U} 's login at \mathcal{S} is not expired yet.

1. s : access source site \mathcal{S} :

- (a) $\mathcal{B} \rightarrow \mathcal{S}$: GET $\langle \text{IST-URL} \rangle_{\mathcal{S}}$? TARGET= $\langle \text{Target} \rangle$... $\langle \text{HTTP-Version} \rangle$
 - where the $\langle \text{HTTP-Version} \rangle$ is chosen by \mathcal{B} as a valid HTTP version String according to [RFC2145].

Constraints:

- (a) [CP1a]: Confidentiality and integrity must be maintained [as assumed in the assumptions [AP2] and [AP1]].
- (b) **Note:** According to general constraint [CG3a] a single target description must be included in the SAML searchpart component. The profile does not use the term SAML searchpart here. Therefore the constraint does not meet this step and multiple target descriptions could be possible.

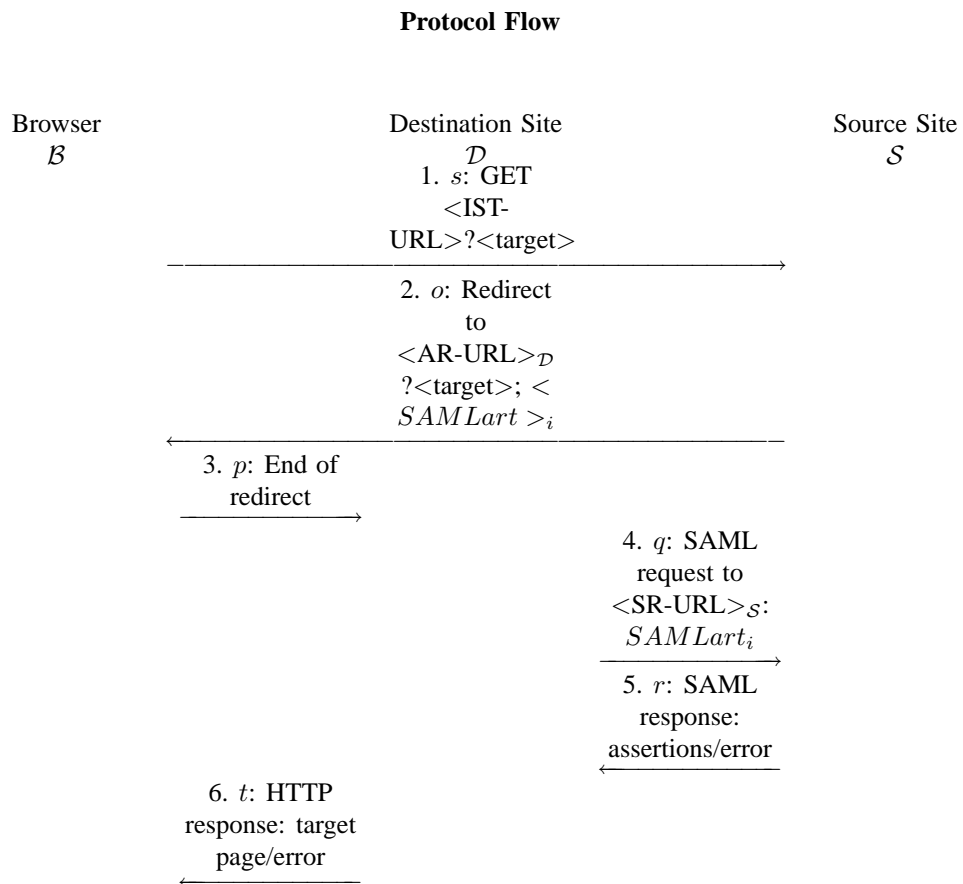


Figure 3.2: Protocol Flow of the SAML Single Sign-on Browser/Artifact Profile

2. *o*: redirect request to *D*:

- (a) *S* : Determines the destination site *D* corresponding to the $\langle \text{Target} \rangle$ of Step 1. Looks up $\langle \text{AR-URL} \rangle_{\mathcal{D}}$ in its artifact receiver table.
- (b) *S* : Generates one or more SAML artifacts $\langle \text{SAMLart} \rangle_i$ according to [CG4] that contain the *SourceID_S*.
- (c) *S* : Generates a SAML searchpart *SAMLsp* according to Definition 5.
- (d) $S \rightarrow \mathcal{B} : \langle \text{HTTP-Version} \rangle 302 \langle \text{Reason Phrase} \rangle$
Location $\langle \text{AR-URL} \rangle_{\mathcal{D}}? \text{SAMLsp}$
 - where the $\langle \text{HTTP-Version} \rangle$ is chosen by *S* as a valid HTTP version String according to [RFC2145].
 - where $\langle \text{Reason Phrase} \rangle$ is a String redirect reason description.

Constraints:

- (a) [CP2a]: Confidentiality and integrity must be maintained [as assumed in the assumptions [AP2] and [AP1]].
- (b) The searchpart *SAMLsp* used must have the format given in the general constraint [CG3]:
 - A single target description $\langle \text{Target} \rangle$ must be included in the SAML searchpart component *SAMLsp*.
 - At least one SAML artifact $\langle \text{SAMLart} \rangle_i$ must be included in the searchpart *SAMLsp*.
 - If more than one artifact $\langle \text{SAMLart} \rangle_i$ is carried within the SAML searchpart *SAMLsp*, all artifacts $\langle \text{SAMLart} \rangle_i$ must have the same *SourceID*.

3. *p*: redirect to *D*:

- (a) *B* : Extracts the URL $\langle \text{AR-URL} \rangle_{\mathcal{D}}? \text{SAMLsp}$ from the Location String of redirect request *o*.
- (b) $\mathcal{B} \rightarrow \mathcal{D} : \text{GET } \langle \text{AR-URL} \rangle_{\mathcal{D}}? \text{SAMLsp } \langle \text{HTTP-Version} \rangle$
 - where the $\langle \text{HTTP-Version} \rangle$ is chosen by *B* as a valid HTTP version String according to [RFC2145].
 - where *SAMLsp* is a SAML searchpart according to Definition 5.

Constraints:

- (a) [CP3a]: Confidentiality and integrity must be maintained [as assumed in the assumptions [AP2] and [AP1]].
- (b) The SAML searchpart *SAMLsp* must have the format given in the general constraint [CG3]:
 - A single target description [$\langle \text{Target} \rangle$] must be included in the SAML searchpart component [*SAMLsp*].
 - At least one SAML artifact [$\langle \text{SAMLart} \rangle_i$] must be included in the searchpart [*SAMLsp*].
 - If more than one artifact [$\langle \text{SAMLart} \rangle_i$] is carried within the SAML searchpart [*SAMLsp*], all the artifacts [$\langle \text{SAMLart} \rangle_i$] must have the same *SourceID*.

4. *q*: SAML request to *S*:

- (a) *D* : Check that each artifact in *p* contains the same *SourceID*.

- (b) \mathcal{D} : Lookup $\langle \text{SR-URL} \rangle_{\mathcal{S}}$ corresponding to SourceID in $\langle \text{SAMLart} \rangle_i$ according to [CP4c]
- (c) Generate RequestID according to [CG1] and [CG2].
- (d) $\mathcal{D} \rightarrow \mathcal{S}$: SAML request to $\langle \text{SR-URL} \rangle_{\mathcal{S}}$
 - with $\langle \text{SAMLart} \rangle_i$ for $i \in \{0, \dots, n-1\}$
 - which has to fulfill [CG1], [CG2] and [CP4d].

Constraints:

- (a) [CP4a]: Confidentiality, integrity and bilateral authentication must be maintained [as assumed in the assumptions [AP2] and [AP1]]. [SAML02b, p. 16]
- (b) The RequestID must fulfill the general constraints [CG1] and [CG2]:
 - The [destination site \mathcal{D}] that assigns [the identifier RequestID] must ensure that there is a negligible probability that [the destination site \mathcal{S}] or any other party will accidentally assign the same identifier to a different data object.
 - Where the [SAML request] declares that it has a particular identifier [RequestID], there must be exactly one such declaration.
 - The values of the RequestID attribute in [the] request [q of Step 4] and the InResponseTo attribute in the corresponding response [r of Step 5] must match.
- (c) [CP4c]: On receiving the SAML artifact [in the redirect p], the destination site determines if the SourceID belongs to a known source site and obtains the site location [and $\langle \text{SR-URL} \rangle$] before sending a SAML request [q]. [SAML02b, p. 17]
- (d) [CP4d]: The destination site must send a SAML request message [q] to the source site, requesting assertions by supplying assertion artifacts in the AssertionArtifact element. [SAML02b, p. 16]

5. r : SAML response to \mathcal{D} :

- (a) \mathcal{S} : Check for artifact destination equality ([CP5j]).
- (b) \mathcal{S} : lookup or generation of SAML assertions corresponding to $\langle \text{SAMLart} \rangle_i$ for $i \in \{0, \dots, n-1\}$ according to [CP5d].
- (c) \mathcal{S} : Generate ResponseID according to [CG1].
- (d) $\mathcal{S} \rightarrow \mathcal{D}$: SAML response from $\langle \text{SR-URL} \rangle_{\mathcal{S}}$
 - with assertions about id_U or error code.
 - which fulfills [CP5d] and [CP5f].
 - with a IDReference to the RequestID of q in the InResponseTo element.

Constraints:

- (a) [CP5a]: Confidentiality, integrity and bilateral authentication must be maintained [as assumed in the assumptions [AP2] and [AP1]]. [SAML02b, p. 16]
- (b) The RequestID must fulfill the general constraints [CG1] and [CG2]:
 - The [destination site \mathcal{D}] that assigns [the identifier RequestID] must ensure that there is a negligible probability that [the destination site \mathcal{D}] or any other party will accidentally assign the same identifier to a different data object.

- Where the [SAML request] declares that it has a particular identifier [RequestID], there must be exactly one such declaration.
 - The values of the RequestID attribute in [the] request [q of Step 4] and the InResponseTo attribute in the corresponding response [r of Step 5] must match.
 - If the RequestID of [the] request [q of Step 4] cannot be determined, then the InResponseTo of the response [r] must not be present. Otherwise it must be present and match the value of the corresponding RequestID attribute [of the SAML request q of Step 4].
- (c) The ResponseID must fulfill the general constraint [CG1]:
- The [source site \mathcal{S}] that assigns [the identifier ResponseID to r] must ensure that there is a negligible probability that [the source site \mathcal{S}] or any other party will accidentally assign the same identifier to a different data object.
 - Where the [SAML response r] declares that it has a particular identifier [ResponseID], there must be exactly one such declaration.
- (d) [CP5d]: If the source site [\mathcal{S}] is able to find or construct the requested assertions, it responds with a SAML response message [r] with the requested assertions. Otherwise it returns an appropriate error code. [SAML02b, p. 16]
- (e) [CG5e]: At least one of the SAML assertions returned to the destination site MUST be a SSO assertion. [SAML02b, p. 16]
- (f) [CP5f]: In the case where the source site returns assertions within SAML response [r], it must return exactly one assertion for each SAML artifact found in the corresponding SAML request element [q of Step 4]. [SAML02b, p. 16]
- (g) For each SAML assertion included into r must hold that its AssertionID fulfills the constraint [CG1]:
- The [source site \mathcal{S}] that assigns [the identifier AssertionID to the assertion] must ensure that there is a negligible probability that [the source site \mathcal{S}] or any other party will accidentally assign the same identifier to a different data object.
 - Where the [SAML assertion] declares that it has a particular identifier [AssertionID], there must be exactly one such declaration.
- (h) [CP5h]: The source site [\mathcal{S}] must implement an one-time request property for each SAML artifact. [SAML02b, p. 16]
- (i) [CP5i]: If a SAML artifact is presented to the source site again, the source site must return the same message as it would if it were queried with an unknown artifact. [SAML02b, p. 16]
- (j) [CP5j]: Artifact destination equality. The source site must return a response [r] with no assertions, if it receives a SAML request message [q] from an authenticated destination site \mathcal{X} containing an artifact issued by the source site to some other destination site \mathcal{Y} , where $\mathcal{X} \neq \mathcal{Y}$. [SAML02b, p. 16]

6. Response to \mathcal{B} :

- (a) $\mathcal{D} \rightarrow \mathcal{B}$: Response with error code or target site.

Constraints:

No normative form is mandated for the HTTP response. [SAML02b, p. 17]

Definition 6 (Correspondence of SAML Request and Response) Let \mathcal{D} be an honest destination site. Let \mathcal{S} be a source site with a SAML responder URL $\langle SR\text{-}URL \rangle_{\mathcal{S}}$. The party \mathcal{D} sent a SAML request $q_{\mathcal{D}}$ to $\langle SR\text{-}URL \rangle_{\mathcal{S}}$ in Step 4. Let \mathcal{S}' be a source site, from which \mathcal{D} received the SAML response $r_{\mathcal{S}'}$ in Step 5. Correspondence in the view of the destination site \mathcal{D} of the SAML response $r_{\mathcal{S}'}$ to the SAML request $q_{\mathcal{D}}$ means:

1. The RequestID of $q_{\mathcal{D}}$ equals the RequestID in the InResponseTo element of $r_{\mathcal{S}'}$.
2. \mathcal{D} could verify in Step 5 that \mathcal{S} equals \mathcal{S}' .

Definition 7 Belief by \mathcal{D} to have a secure channel C with a person \mathcal{U} with name $id_{\mathcal{U}}$ according to the SAML SSO browser/artifact profile means:

1. Step 3 redirect: Destination site \mathcal{D} received a redirect p with one or more SAML artifacts $SAMLart_i^{(\mathcal{D})}$ through a secure channel C to its $\langle AR\text{-}URL \rangle_{\mathcal{D}}$.
2. [CP4c] fulfilled: \mathcal{D} looked up the $\langle SR\text{-}URL \rangle_{\mathcal{S}}$ of source site \mathcal{S} corresponding to the artifacts $SAMLart_i$'s sourceIDs.
3. Step 4 SAML request sent: \mathcal{D} sent a SAML request q with request id $RequestID_{\mathcal{D}}$ and containing the artifacts $SAMLart_i$ to the $\langle SR\text{-}URL \rangle_{\mathcal{S}}$ source site \mathcal{S} .
4. Step 5 finished successful: \mathcal{D} accepted a SAML response r from the $\langle SR\text{-}URL \rangle_{\mathcal{S}}$ of \mathcal{S} with the chosen $RequestID_{\mathcal{D}}$ in the InResponseTo element. The SAML response r has to fulfill the constraints given in Step 5 and contain at least one SSO assertion with $id_{\mathcal{U}}$ in the NameIdentifier.

3.2.6 State Model for the Protocol

In this section we depict the protocol states with different diagrams. The diagrams are deduced from the protocol description and not base of the security proof of the SAML Single Sign-on Browser/Artifact Profile. They are supposed to show the protocol flows from another point of view and serve a better understanding.

Collaboration Diagram of the Participating Parties

In Figure 3.3 we show a collaboration diagram of the different parties involved in the protocol. We depict the parties as interacting state machines as suggested in [Pfi99]. The ellipses on the lower borders symbolize input/output ports of a party, that ones in the upper borders the user interfaces.

State Machines

In this section we provide the state charts of the machines presented in Section 3.2.6. **Note:** The sub-states of different states represent conditions that have to be tested or actions to be taken. If a state contains a set of sub-states, this says that all these actions have to be taken and does not imply a certain sequence of execution.

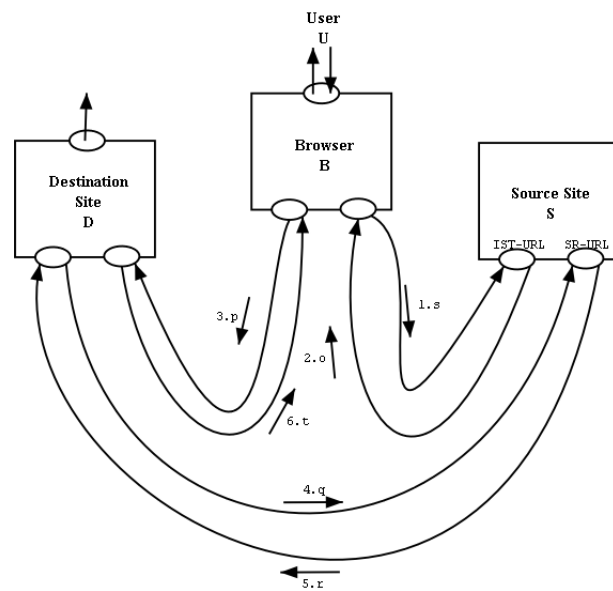


Figure 3.3: Collaboration Diagram of the Interacting State Machines

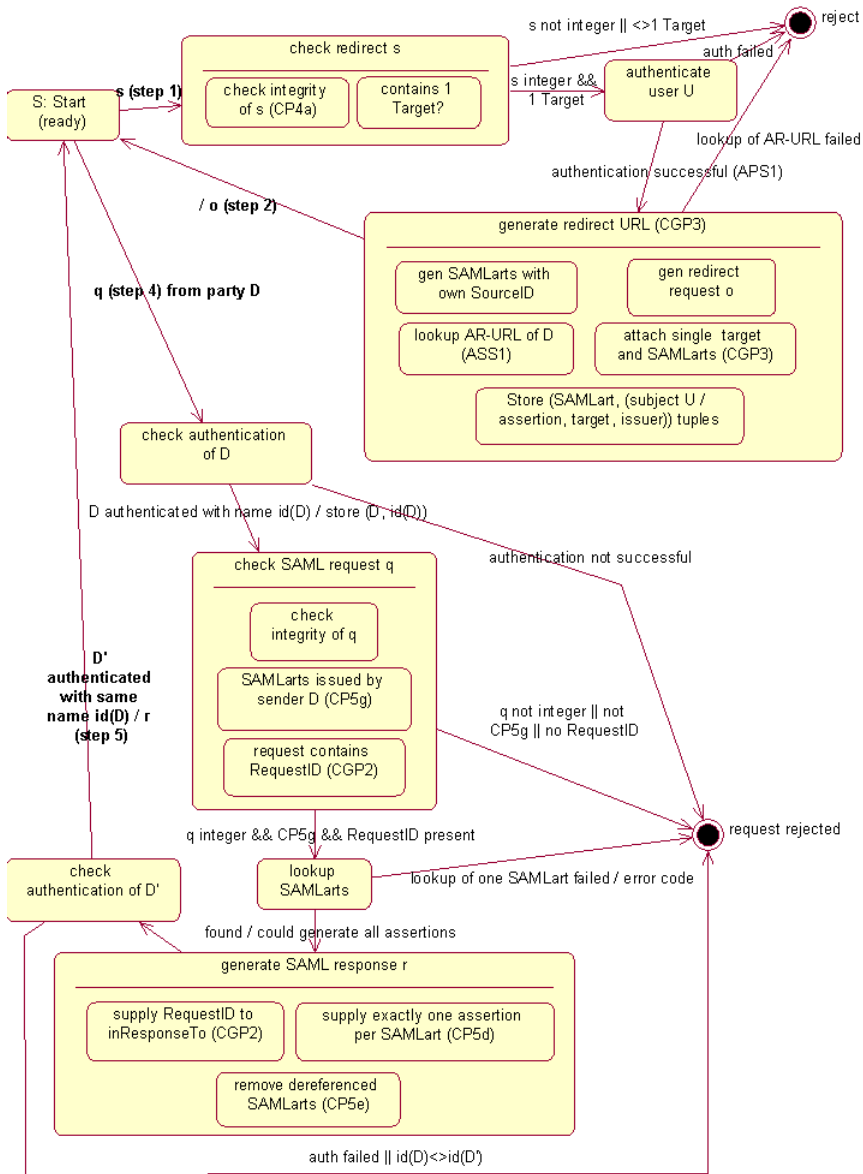


Figure 3.4: State Chart of the Source Site S

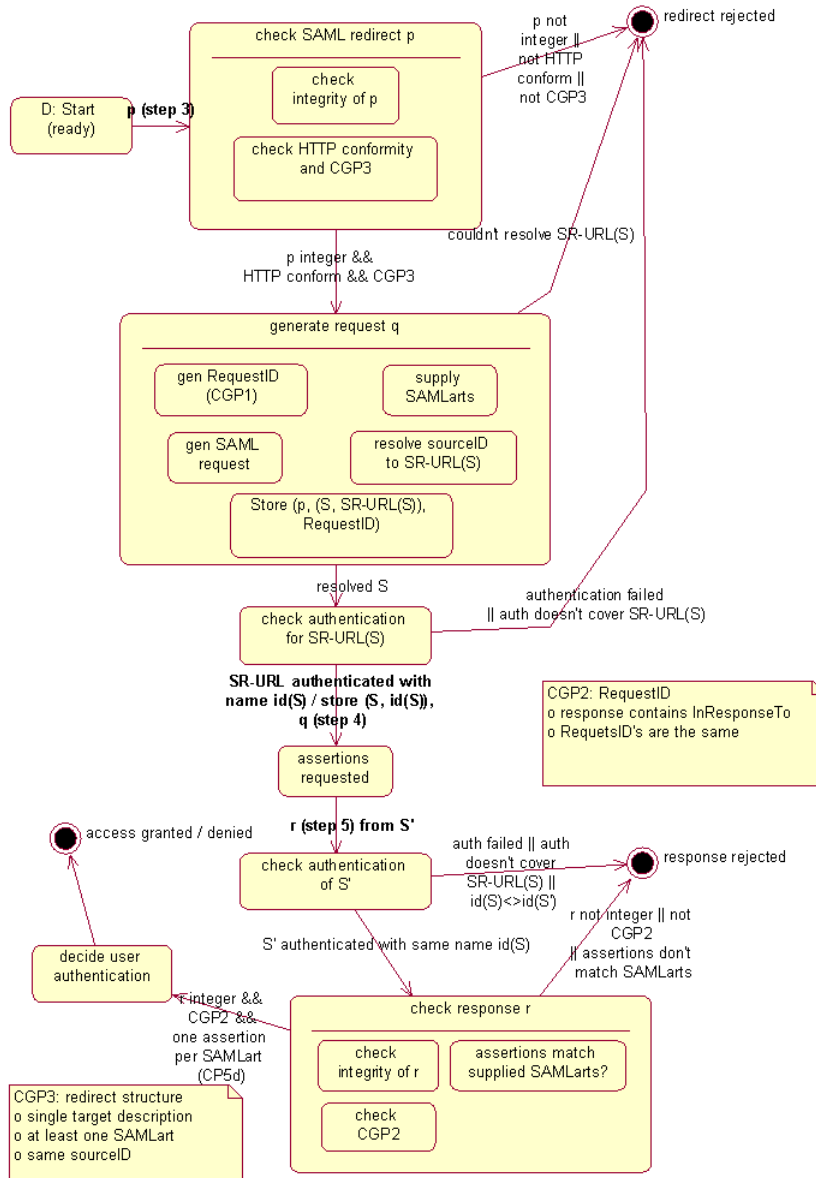
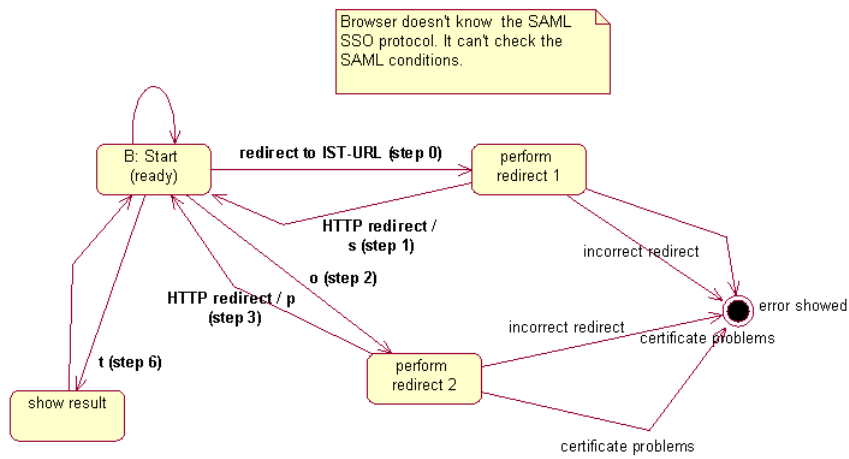


Figure 3.5: State Chart of the Destination Site \mathcal{D}

Figure 3.6: State Chart of the Browser \mathcal{B}

3.3 Assumptions

In this section we present the assumptions we made about the setup of the protocol (Section 3.3.1) and the protocol run itself (Section 3.3.2). The SAML Single Sign-on Profile itself describes further more detailed constraints of the protocol run. You find these constraints in the different protocol steps of Section 3.2.5.

3.3.1 Assumptions about the Setup

1. **[ASS1]**: An honest source site \mathcal{S} knows the tuple of identity and artifact receiver URL ($id_{\mathcal{D}}$, $\langle AR-URL \rangle_{\mathcal{D}}$) of each destination site \mathcal{D} that it is supposed to communicate with beforehand. This table is transferred out of the line.
1. **[ASD1]**: An honest destination site \mathcal{D} possesses a table with triples of certified identities, valid SourceIDs and corresponding SAML responder URLs ($id_{\mathcal{S}}$, $SourceID_{\mathcal{S}}$, $\langle SR-URL \rangle_{\mathcal{S}}$) of non-local source sites \mathcal{S} . This table is transferred out of the line and only contains $SourceIDs$ of honest source sites. **Note:** This assumption is based on the requirements given for the SAML artifact format [SAML02b, p. 17]. The browser/artifact profile says about the SourceID of the SAML artifact: It is assumed that the destination site will maintain a table of SourceID values as well as the URL (or address) for the corresponding SAML responder. This information is communicated between the source and destination sites out of band.

3.3.2 Assumptions about the Protocol Run

1. **[AP1]**: Let \mathcal{X} be the sender of a message m as described in a protocol step and \mathcal{Y} the intended receiver. The integrity property implies existential unforgeability under adaptive chosen-message attack. The receiver \mathcal{Y} either receives the message m as sent by \mathcal{X} or gets a failure message. \mathcal{X} in another connection only with negligible probability.²
2. **[AP2]**: Let \mathcal{X} be the sender of a message m as described in a protocol step and \mathcal{Y} the intended receiver. We interpret the confidentiality property needed in the protocol steps as semantic security of m under adaptive chosen-message attack. Other parties than \mathcal{X} and \mathcal{Y} cannot guess any function of m with non-negligible probability.

3.4 Attacks on the Unmodified Profile

In this section we provide different attacks on the SAML Single Sign-on Browser/Artifact profile. Some of them rely on unclear specifications in the profile and interpret them in a ridiculous way which makes the attack possible. Other attacks show vulnerabilities in the abstract profile.

Certain SAML bindings are resistant against these attacks because they provide additional authentication or robustness measures, where the profile itself does not claim it. One example is the SOAP over HTTP binding that uses SSL/TLS as secure channel.

²This assumption matches symmetric authentication codes as well as asymmetric signatures. With symmetric authentication you will have a collision of two symmetric keys in different connections only with negligible probability. With signatures you have a long term link to the signing party.

In Section 3.4.1 we show a replay attack on the destination site. In Section 3.4.2 we provide two different examples for man-in-the-middle attacks on the profile. Section 3.4.3 contains an attack based on information leakage through the HTTP referer tag.

[KR00] describes another attack that is possible on the SAML SSO profile, the so-called Bogus Merchant attack. It was originally used as attack on Microsoft Passport [Mic02a] and based on the weakness of the certification authorities' signing policy. This attacks is outside of the scope of this thesis.

3.4.1 Connection Hijacking / Replay Attack

An adversary can break the SAML Single Sign-on Profile by connection hijacking and replay of an encrypted redirect. The used technique of connection hijacking is described in [DM02]. This is possible under the following prerequisites:

Prerequisites

- The adversary \mathcal{A} is capable of connection interception and hijacking.
- The adversary \mathcal{A} can observe the connection from the browser \mathcal{B} to artifact receiver URL $\langle \text{AR-URL} \rangle_{\mathcal{D}}$ of Step 3.
- The HTTP request of Step 3 is transported through a TCP/IP connection or any other connection the adversary \mathcal{A} can intercept.
- The integrity property claimed in Step 3 is interpreted as message integrity without binding to the sending party.

The Attack The attack refers to the protocol schema in Section 3.2.5 on Page 28.

3. $\mathcal{B} \rightarrow \mathcal{D}$: Redirect p to $\langle \text{AR-URL} \rangle_{\mathcal{D}}$
 \mathcal{A} : The adversary \mathcal{A} intercepts this redirect and finishes the connection from \mathcal{B} to \mathcal{D} .
- 3*. $\mathcal{A}_{\mathcal{B}} \rightarrow \mathcal{D}$: replay of redirect p to $\langle \text{AR-URL} \rangle_{\mathcal{D}}$
 Because of the integrity and confidentiality property of Step 3 adversary \mathcal{A} cannot modify the redirect message or read it in plain text. He resends the message as seen in the previous step impersonating the browser \mathcal{B} to \mathcal{D} .
 The destination site cannot distinguish the browser \mathcal{B} from \mathcal{A} because of the lack of authentication and therefore proceeds as specified in the protocol.
4. $\mathcal{D} \rightarrow \mathcal{S}$: SAML request q with SAML artifacts of p .
5. $\mathcal{S} \rightarrow \mathcal{D}$: SAML response r with assertions.
 The artifacts given in the SAML request q of \mathcal{D} were issued to \mathcal{D} . The SAML request is identical to that one \mathcal{D} would have sent, if it were connected to \mathcal{B} .
- 6*. $\mathcal{D} \rightarrow \mathcal{A}_{\mathcal{B}}$: Response to Step 3*
 \mathcal{D} will respond to $\mathcal{A}_{\mathcal{B}}$ and grant $\mathcal{A}_{\mathcal{B}}$ the same permissions as \mathcal{B} with user \mathcal{U} and certified identity $id_{\mathcal{U}}$

Why does this attack work? Because of the integrity and confidentiality property of Step 3 the adversary \mathcal{A} cannot see or modify the content of the message p . Unfortunately these properties are not strong enough to prevent a replay attack. The redirect message p does not contain any short-term freshness measures.

The adversary \mathcal{A} impersonates the browser \mathcal{B} to the destination site \mathcal{D} . Because of the lack of authentication in this step and missing identifiers in the redirect, the destination site \mathcal{D} cannot distinguish between the parties \mathcal{B} and \mathcal{A} or the redirect messages $p_{\mathcal{B}}$ and $p_{\mathcal{A}}$. Apart from the IP address of the communication partner the view of \mathcal{D} is the same in the communication with \mathcal{B} and $\mathcal{A}_{\mathcal{B}}$.

Protocol Flow

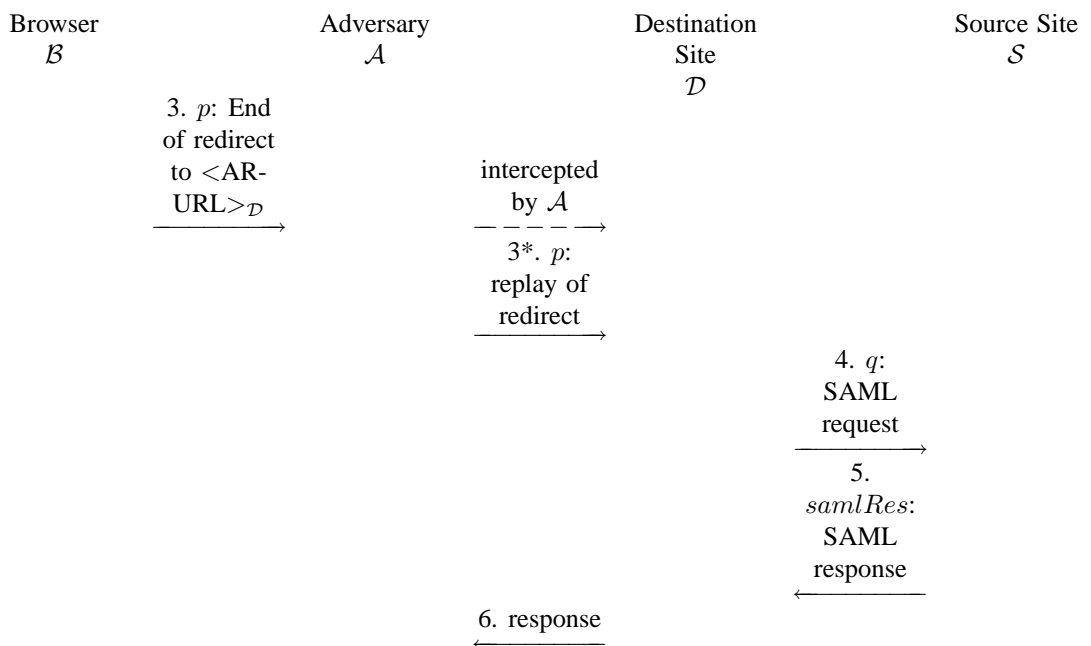


Figure 3.7: Part of the Protocol Flow Showing the Hijacking Attack

Possible solutions

- Only as heuristic: \mathcal{S} puts the IP address of browser \mathcal{D} in the redirect, equality check by \mathcal{D} .
- Integrity property has to include binding to the sending party or the underlying channel.
- Secure channel $\mathcal{B} \leftrightarrow \mathcal{D}$ in steps 3 and 6, which provides freshness and replay prevention.

3.4.2 Man-in-the-Middle Attacks

In this section we consider two different examples of man-in-the-middle attacks. The general technique of man-in-the-middle attacks is described in [Bha01]. [Mea96]

provides a concrete example for such an attack on the Needham-Schroeder Public-Key protocol. Our attacks rely on different assumptions and take place in different parts of the protocol.

Between \mathcal{B} and \mathcal{D} by DNS Spoofing

The presented attack uses the well known weakness that an adversary who controls the Domain Name Service (DNS) can impersonate one party to another. We present a man-in-the-middle attack, whereas the adversary \mathcal{A} is a proxy between browser \mathcal{B} and destination site \mathcal{D} : $\mathcal{B} \leftrightarrow \mathcal{A} \leftrightarrow \mathcal{D}$. An adversary can get also man-in-the-middle between \mathcal{S} and \mathcal{B} by DNS spoofing.

Prerequisites

- Let \mathcal{A} be an adversary, who can break DNS. That means he can impersonate certain URLs to other parties.
- We assume that there is no authentication in Step 3 as stated in the browser/artifact profile.

The Attack The attack refers to the protocol schema in Section 3.2.5 on Page 28.

$\mathcal{A}_{\mathcal{D}}$: Impersonates the destination site's $\langle \text{AR-URL} \rangle_{\mathcal{D}}$ to browser \mathcal{B} . \implies
 $\langle \text{AR-URL} \rangle_{\mathcal{D}}^{(\mathcal{B})} = \langle \text{AR-URL} \rangle_{\mathcal{A}}$.

3. $\mathcal{B} \rightarrow \mathcal{A}_{\mathcal{D}}$: redirect p with artifacts to $\langle \text{AR-URL} \rangle_{\mathcal{D}}^{(\mathcal{B})}$
 Because the DNS corruption the browser \mathcal{B} connects to $\langle \text{AR-URL} \rangle_{\mathcal{A}}$.

3*: $\mathcal{A}_{\mathcal{B}} \rightarrow \mathcal{D}$: redirect to $\langle \text{AR-URL} \rangle_{\mathcal{D}}$
 \implies Classical man-in-the-middle attack. The destination site \mathcal{D} cannot distinguish $\mathcal{A}_{\mathcal{B}}$ from browser \mathcal{B} , and \mathcal{B} cannot distinguish $\mathcal{A}_{\mathcal{D}}$ from the honest destination site \mathcal{D} .

Why does this attack work? The profile does not claim authentication in Step 3. Therefore the browser \mathcal{B} cannot distinguish the $\langle \text{AR-URL} \rangle_{\mathcal{A}}$ of the adversary $\mathcal{A}_{\mathcal{D}}$ from $\langle \text{AR-URL} \rangle_{\mathcal{D}}$ that belongs to the honest destination site \mathcal{D} . Because of the same fact the honest destination site \mathcal{D} cannot distinguish an honest browser \mathcal{B} from the adversary $\mathcal{A}_{\mathcal{B}}$.

Possible Solutions

- Unilateral authentication: The destination site has to authenticate to browser \mathcal{B} .
- Secure channel $\mathcal{B} \leftrightarrow \mathcal{D}$ in steps 3 and 6.

Between \mathcal{B} and \mathcal{S} by Message Rewriting

We present another man-in-the-middle attack, whereas the adversary uses the possibility to rewrite the target of the redirect in Step 1. He works as proxy between \mathcal{B} and \mathcal{S} : $\mathcal{B} \leftrightarrow \mathcal{A} \leftrightarrow \mathcal{S}$.

Prerequisites

- Let \mathcal{A} be an adversary that is capable of rewriting HTTP messages.
- Let the method of tracking an authenticated user of source site \mathcal{S} be unprotected against man-in-the-middle attacks.

The Attack The attack refers to the protocol schema in Section 3.2.5 on Page 28.

0. $\mathcal{P} \rightarrow \mathcal{B}$: redirect request to $\langle \text{IST-URL} \rangle_{\mathcal{S}}$
 whereas \mathcal{P} initiates the redirect to $\langle \text{IST-URL} \rangle_{\mathcal{S}}$ of honest source site \mathcal{S} .

\mathcal{A} : Rewrite of redirect request

The profile does not state anything about the previous step. Therefore we cannot assume confidentiality or integrity for that connection. \mathcal{A} rewrites the redirect request such that it redirects to $\langle \text{IST-URL} \rangle_{\mathcal{A}}$, a bogus inter-site transfer URL of his own.

1. $\mathcal{B} \rightarrow \mathcal{A}_{\mathcal{S}}$: finish of the redirect
 Please note that the profile does not claim authentication for this step.
- 1*. $\mathcal{A}_{\mathcal{B}} \rightarrow \mathcal{S}$: finish of the redirect
 The adversary $\mathcal{A}_{\mathcal{B}}$ impersonates the browser \mathcal{B} to \mathcal{S} .

\mathcal{A} : Works as man-in-the-middle between \mathcal{B} and \mathcal{S} .

Because the user tracking system of \mathcal{S} is not resistant against man-in-the-middle attacks, he can forward all communication between \mathcal{B} and \mathcal{S} during the user tracking.

- 2*. $\mathcal{S} \rightarrow \mathcal{A}_{\mathcal{B}}$: redirect request to $\langle \text{AR-URL} \rangle_{\mathcal{D}}$
 This redirect contains the SAML artifacts for \mathcal{B} readable by $\mathcal{A}_{\mathcal{B}}$. $\mathcal{A}_{\mathcal{B}}$ now begins to impersonate \mathcal{B} to destination site \mathcal{D} .
- 3*. $\mathcal{A}_{\mathcal{B}} \rightarrow \mathcal{D}$: finish of redirect to $\langle \text{AR-URL} \rangle_{\mathcal{D}}$
 The adversary $\mathcal{A}_{\mathcal{B}}$ impersonates \mathcal{B} to \mathcal{D} . The redirect contains valid SAML artifacts and allows $\mathcal{A}_{\mathcal{B}}$ to act with the permissions of \mathcal{U} .
2. $\mathcal{A} \rightarrow \mathcal{B}$: redirect request to $\langle \text{IST-URL} \rangle_{\mathcal{S}}$
 The adversary \mathcal{A} sends the original redirect request of Step 0 to the browser \mathcal{B} . As the profile assumes that the user already is authenticated to the source site \mathcal{S} , there need not be any user interaction between the protocol steps 1 and 2. Therefore the adversary can reinitiate a normal protocol run of \mathcal{B} with a high probability that the user would not notice the reset of the protocol run.

Why does this attack work? The SAML single sign-on browser/artifact profile does not address the initialization of the redirect in Step 1 (Such to say Step 0). Therefore we cannot claim integrity for this step and the adversary \mathcal{A} can rewrite the target URL of the redirect. Because of the lack of authentication in the steps 1 and 2 the adversary \mathcal{A} can work as man-in-the-middle between \mathcal{B} and \mathcal{S} . The profile does not state, which security assumptions can be made about the tracking of an authenticated user by source site \mathcal{S} . Therefore we can assume that the adversary can forward this communication, too.

Possible solutions

- Unilateral authentication in all protocol steps.
- Strong tracking of authenticated users.

3.4.3 HTTP Referer Attack

The following attack allows an adversary \mathcal{A} provoke an information leakage of valid SAML artifacts. It uses the Referer Tag of HTTP (see [RFC2616]) to get hold of unused SAML artifacts.

Prerequisites

- Let \mathcal{A} be an adversary, who can tap all channels and intercept arbitrary connections.
- Let \mathcal{B} be a browser, which sets the Referer Tag by default.
- Either the error page the destination site \mathcal{D} generates in case of a protocol failure contains a link to an URL not providing confidentiality or integrity. Or the adversary \mathcal{A} is able to manipulate data transferred through connections that do not maintain the integrity property.

The Attack The attack refers to the protocol schema in Section 3.2.5 on Page 28.

3. $\mathcal{B} \rightarrow \mathcal{D}$: redirect to $\langle \text{AR-URL} \rangle_{\mathcal{D}}$
which contains valid SAML artifacts
4. $\mathcal{D} \rightarrow \mathcal{S}$: SAML request to $\langle \text{SR-URL} \rangle_{\mathcal{S}}$
 \mathcal{A} : The adversary \mathcal{A} intercepts this message. Therefore the SAML request is without success. Therefore \mathcal{D} sends an error message in the HTTP response of Step 6.
6. $\mathcal{D} \rightarrow \mathcal{B}$: error message
According to the profile in this or following steps no confidentiality or integrity is required. The adversary \mathcal{A} can proceed in two ways:
 - If the error message contains a link or redirect to an URL to party \mathcal{P} which is not secured, the adversary \mathcal{A} can tap the channel the browser eventually builds up to \mathcal{P} .
 - If the adversary \mathcal{A} is able to manipulate data transferred through connections, he can change the message of Step 6 to a redirect to a party \mathcal{P} chosen by him.
7. $\mathcal{B} \rightarrow \mathcal{P}$: HTTP request
Next request of the browser \mathcal{B} . Contains $\langle \text{AR-URL} \rangle_{\mathcal{D}}$ in the referer tag including the query string and therefore the still valid SAML artifacts of Step 4. \mathcal{A} can read them in plain text.
- 3*. $\mathcal{A} \rightarrow \mathcal{D}$: redirect to $\langle \text{AR-URL} \rangle_{\mathcal{D}}$
as read in the referer tag.

Why does this attack work? The message of Step 6 allows the possibility of information leakage: It does not claim confidentiality oder integrity. The referer tag is normally set, if the source the user comes from, has an own URI. It will contain the redirect to the $\langle \text{AR-URL} \rangle_{\mathcal{D}}$ which contains the still valid SAML artifacts.

Possible solutions

- Dereferer redirect before Step 6. Only needed if Step 4 or 5 were not successful or not all artifacts were consumed by \mathcal{D} .
- Secure channel between browser and destination site in Step 3 kept alive for the response in Step 6.

Chapter 4

Repair of the SAML Single Sign-on Profile

We enhance the SAML Single Sign-on Browser/Artifact Profile to make it resistant against the described attacks. On the one hand we extend the protocol by robustness measures common in in protocol design. On the other hand we take additional assumptions that provide security against the described attacks.

4.1 Proposed Changes

We make different changes to the SAML Single Sign-on Profile. They are supposed to enhance general security and to prevent the attacks shown in Section 3.4.

Preciseness: Within the SAML Single Sign-on Browser/Artifact Profile the searchpart of Step 1 is not considered as SAML searchpart. Thus, the format description and security rules for the SAML searchpart do not imply anything for Step 1. We consider the searchpart in each URL used within the profile as SAML searchpart and specify the differences in the each step explicitly.

We explicitly name the tables that the different parties store.

Robustness: We enhance the profile with robustness measures in the SAML searchpart as proposed by [Pfi01]. We include an explicit naming of protocol and protocol step into the URL. As SAML is an emerging technology, it is likely that there are different protocols with similar messages. This naming prevents an adversary to use messages of SAML SSO in other steps as intended or within other protocols (and vice versa).

Stronger Authentication: We expect stronger authentication measures of the participating parties. We require unilateral authentication in the steps 1-3. This hardens man-in-the-middle attacks.

Stronger User Tracking: We also require a user tracking that is secure against those attacks. The term user tracking refers to the capability of a source site to recognize a user that previously logged in. There are different techniques on the market, where some are secure against man-in-the-middle attacks (for instance by utilizing SSL channels and special cookies) and some not.

Dereferer Redirect: The original profile is vulnerable against information leakage by an attack utilizing the referer tag. We require to prevent adversaries from collecting information of the referer tag.

4.2 Improved Profile

Please note that we still use the general constraints of Section 3.2.5 and the new constraints of the different protocol steps are still compliant to given SAML bindings (i.e. SOAP over HTTP). We enhance the SAML searchpart as follows

Definition 8 (Enhanced SAML searchpart) We call a searchpart of an URL enhanced SAML searchpart, if it contains the following key/value pairs:

- exactly one protocol identifier with key *PROTOCOL* and a value $\langle \text{ProtocolID} \rangle$ equal to the SAML single sign-on browser/artifact protocol ID.
- exactly one protocol step identifier with key *SSOSTEP* and a value $\langle \text{Step} \rangle \in \{\text{to_IST-URL}, \text{to_AR-URL}\}$.
- exactly one target description with key *TARGET* and a string value $\langle \text{Target} \rangle$, which describes the target.
- zero or more SAML artifacts with key *SAMLart* and a value $\langle \text{SAMLart} \rangle_i$ in the SAML artifact format.

It has the following format:

Enhanced SAML searchpart: ... *TARGET*= $\langle \text{Target} \rangle$...
PROTOCOL= $\langle \text{ProtocolID} \rangle$... *SSOSTEP*= $\langle \text{Step} \rangle$...
SAMLart= $\langle \text{SAMLart} \rangle_0$... *SAMLart*= $\langle \text{SAMLart} \rangle_{n-1}$...

- where “*TARGET*”, “*SAMLart*”, “*PROTOCOL*” and “*SSOSTEP*” are String constants,
- $n \in \mathbb{N}$ is the number of artifacts,
- $\langle \text{ProtocolID} \rangle$ is a String protocol identifier,
- $\langle \text{Step} \rangle$ is a String identifier of the protocol step,
- $\langle \text{Target} \rangle$ is a String description of the target resource and
- each $\langle \text{SAMLart} \rangle_i$ is a SAML artifact.

Protocol Setup

The improved SAML Single Sign-on profile requires the same setup as the original profile. A destination site additionally chooses a URL $\langle \text{DEREF} \rangle_{\mathcal{D}}$, which is used as dereferer URL.

Subprotocol: sso_login

Unchanged.

Subprotocol: single_sign_on

Since the original profile does not define a message before Step 1, we do not fix it either. The SAML SSO needs this flexibility to be usable in portal scenarios, where a web portal forwards the browser to the user's source site. Our improved profile also starts with a redirect s to the $\langle\text{IST-URL}\rangle_{\mathcal{S}}$ of a source site \mathcal{S} . As in the original profile another party provides the $\langle\text{IST-URL}\rangle_{\mathcal{S}}$ of source site \mathcal{S} and a String target description $\langle\text{Target}\rangle$. The enhanced searchpart $SAMLsp_{enh}$ contains this target description that refers to a resource on a destination site \mathcal{D} That the user \mathcal{U} wants to browse.

Precondition: The user \mathcal{U} and a source site \mathcal{S} have run the sso_login sub-protocol: The user \mathcal{U} logged in at source site \mathcal{S} successfully. \mathcal{U} provided the login information $login_{\mathcal{U}}$ via \mathcal{U} 's browser \mathcal{B} . The source site \mathcal{S} verified this login information and returned $id_{\mathcal{U}}$. The validity time of \mathcal{U} 's login at \mathcal{S} is not expired yet.

1. s : access source site \mathcal{S} :

- (a) $\mathcal{B} \rightarrow \mathcal{S}$: GET $\langle\text{IST-URL}\rangle_{\mathcal{S}}? SAMLsp_{enh} \dots \langle\text{HTTP-Version}\rangle$
 - where $SAMLsp_{enh}$ is a SAML searchpart according to Definition 8 with $SSOSTEP=to_IST-URL$ and no SAML artifacts.
 - where the $\langle\text{HTTP-Version}\rangle$ is chosen by \mathcal{B} as a valid HTTP version String according to [RFC2145].
- (b) $\mathcal{B} \leftrightarrow \mathcal{S}$: Secure tracking of user \mathcal{U} .

Constraints:

- (a) **[CP1a]:** Confidentiality, integrity [and unilateral authentication] must be maintained [as assumed in the assumptions **[AP2]** and **[AP1]**].
- (b) The URL used must have the format given in Definition 8 and fulfill:
 - A single target description $\langle\text{Target}\rangle$ must be included in the enhanced SAML searchpart component $SAMLsp_{enh}$.
 - The searchpart $SAMLsp_{enh}$ contains no SAML artifact.
- (c) The tracking of user \mathcal{U} is secured against man-in-the-middle attacks.

2. o : redirect request to \mathcal{D} :

- (a) \mathcal{S} : Determines the destination site \mathcal{D} corresponding to the $\langle\text{Target}\rangle$ of Step 1. Looks up $\langle\text{AR-URL}\rangle_{\mathcal{D}}$ in its artifact receiver table.
- (b) \mathcal{S} : Generates one or more SAML artifacts $\langle\text{SAMLart}\rangle_i$ according to **[CG4]** that contain the $SourceID_{\mathcal{S}}$.
- (c) \mathcal{S} : Generates a enhanced SAML searchpart $SAMLsp_{enh}$ according to Definition 5 with $SSOSTEP=to_AR-URL$ and and one or more SAML artifacts.
- (d) $\mathcal{S} \rightarrow \mathcal{B}$: $\langle\text{HTTP-Version}\rangle$ 302 $\langle\text{Reason Phrase}\rangle$
Location $\langle\text{AR-URL}\rangle_{\mathcal{D}}? SAMLsp_{enh}$
 - where the $\langle\text{HTTP-Version}\rangle$ is chosen by \mathcal{S} as a valid HTTP version String according to [RFC2145].
 - where $\langle\text{Reason Phrase}\rangle$ is a String redirect reason description.

Constraints:

- (a) **[CP2a]**: Confidentiality and integrity [and unilateral authentication] must be maintained [as assumed in the assumptions **[AP2]** and **[AP1]**].
- (b) The enhanced SAML searchpart $SAMLsp_{enh}$ must have the format given in Definition 8 and fulfill the general constraint **[CG3]**:
 - A single target description $\langle Target \rangle$ must be included in the enhanced SAML searchpart component $SAMLsp_{enh}$.
 - At least one SAML artifact $\langle SAMLart \rangle_i$ must be included in the searchpart $SAMLsp_{enh}$.
 - If more than one artifact $\langle SAMLart \rangle_i$ is carried within the SAML searchpart $SAMLsp_{enh}$, all the artifacts $\langle SAMLart \rangle_i$ must have the same $SourceID$.

3. p : **redirect to \mathcal{D}** :

- (a) \mathcal{B} : Extracts the URL $\langle AR-URL \rangle_{\mathcal{D}}? SAMLsp_{enh}$ from the Location String of redirect request o .
- (b) $\mathcal{B} \rightarrow \mathcal{D}$: GET $\langle AR-URL \rangle_{\mathcal{D}}? SAMLsp_{enh} \langle HTTP-Version \rangle$
 - where the $\langle HTTP-Version \rangle$ is chosen by \mathcal{B} as a valid HTTP version String according to [RFC2145].
 - where $SAMLsp_{enh}$ is a SAML searchpart according to Definition 8 with $SSOSTEP=to_AR-URL$ and and one or more SAML artifacts.

Constraints:

- (a) **[CP3a]**: Confidentiality and integrity must [and unilateral authentication] be maintained [as assumed in the assumptions **[AP2]** and **[AP1]**]. We name the channel used in this connection C .
- (b) The channel C is kept alive until Step 6
- (c) The The enhanced SAML searchpart $SAMLsp_{enh}$ must have the format given in Definition 8 and fulfill the general constraint **[CG3]**:
 - A single target description [$\langle Target \rangle$] must be included in the enhanced SAML searchpart component [$SAMLsp_{enh}$].
 - At least one SAML artifact [$\langle SAMLart \rangle_i$] must be included in the searchpart [$SAMLsp_{enh}$].
 - If more than one artifact [$\langle SAMLart \rangle_i$] is carried within the enhanced SAML searchpart [$SAMLsp_{enh}$], all the artifacts [$\langle SAMLart \rangle_i$] must have the same $SourceID$.

4. q : **SAML request to \mathcal{S}** :

- (a) \mathcal{D} : Check redirect format according to **[CG3]** and Definition 8.
- (b) \mathcal{D} : Check that each artifact in p contains the same $SourceID$.
- (c) \mathcal{D} : Lookup $\langle SR-URL \rangle_{\mathcal{S}}$ corresponding to $SourceID$ in $\langle SAMLart \rangle_i$ according to **[CP4c]**.
- (d) Generate RequestID according to **[CG1]** and **[CG2]**.
- (e) $\mathcal{D} \rightarrow \mathcal{S}$: SAML request to $\langle SR-URL \rangle_{\mathcal{S}}$
 - with $\langle SAMLart \rangle_i$ for $i \in \{0, \dots, n-1\}$,
 - which has to fulfill **[CG1]**, **[CG2]** and **[CP4d]**.

Constraints:

- (a) **[CP4a]**: Confidentiality, integrity and bilateral authentication must be maintained [as assumed in the assumptions **[AP2]** and **[AP1]**]. [SAML02b, p. 16]
- (b) The RequestID must fulfill the general constraints **[CG1]** and **[CG2]**:
 - The [destination site \mathcal{D}] that assigns [the identifier RequestID] must ensure that there is a negligible probability that [the destination site \mathcal{S}] or any other party will accidentally assign the same identifier to a different data object.
 - Where the [SAML request] declares that it has a particular identifier [RequestID], there must be exactly one such declaration.
 - The values of the RequestID attribute in [the] request [q of Step 4] and the InResponseTo attribute in the corresponding response [r of Step 5] must match.
- (c) **[CP4c]**: On receiving the SAML artifact [in the redirect p], the destination site determines if the SourceID belongs to a known source site [by looking up the SourceID in its SR-URL table] and obtains the site location [and \langle SR-URL \rangle] before sending a SAML request [q]. [SAML02b, p. 17]
- (d) **[CP4d]**: The destination site must send a SAML request message [q] to the source site, requesting assertions by supplying assertion artifacts in the AssertionArtifact element. [SAML02b, p. 16]

5. r : SAML response to \mathcal{D} :

- (a) \mathcal{S} : Check for artifact destination equality (**[CP5j]**).
- (b) \mathcal{S} : lookup or generation of SAML assertions corresponding to \langle SAMLart \rangle_i for $i \in \{0, \dots, n-1\}$ according to **[CP5d]**.
- (c) \mathcal{S} : enforce one-time request property of the artifacts that are used according to (**[CP5h]** and **[CP5i]**).
- (d) \mathcal{S} : Generate ResponseID according to **[CG1]**.
- (e) $\mathcal{S} \rightarrow \mathcal{D}$: SAML response from \langle SR-URL $\rangle_{\mathcal{S}}$
 - with assertions about id_U or error code according.
 - which fulfills **[CP5d]** and **[CP5f]**.
 - with a IDReference to the RequestID of q in the InResponseTo element.
- (f) \mathcal{D} : Check for source site equality.

Constraints:

- (a) **[CP5a]**: Confidentiality, integrity and bilateral authentication must be maintained [as assumed in the assumptions **[AP2]** and **[AP1]**]. [SAML02b, p. 16]
- (b) The RequestID must fulfill the general constraints **[CG1]** and **[CG2]**:
 - The [destination site \mathcal{D}] that assigns [the identifier RequestID] must ensure that there is a negligible probability that [the destination site \mathcal{D}] or any other party will accidentally assign the same identifier to a different data object.
 - Where the [SAML request] declares that it has a particular identifier [RequestID], there must be exactly one such declaration.
 - The values of the RequestID attribute in [the] request [q of Step 4] and the InResponseTo attribute in the corresponding response [r of Step 5] must match.

- If the RequestID of [the] request [q of Step 4] cannot be determined, then the InResponseTo of the response [r] must not be present. Otherwise it must be present and match the value of the corresponding RequestID attribute [of the SAML request q of Step 4].
- (c) The ResponseID must fulfill the general constraint [CG1]:
- The [source site S] that assigns [the identifier ResponseID to r] must ensure that there is a negligible probability that [the source site S] or any other party will accidentally assign the same identifier to a different data object.
 - Where the [SAML response r] declares that it has a particular identifier [ResponseID], there must be exactly one such declaration.
- (d) [CP5d]: If the source site [S] is able to find or construct the requested assertions, it responds with a SAML response message [r] with the requested assertions. Otherwise it returns an appropriate error code. [SAML02b, p. 16]
- (e) [CG5e]: At least one of the SAML assertions returned to the destination site MUST be a SSO assertion. [SAML02b, p. 16]
- (f) [CP5f]: In the case where the source site returns assertions within SAML response [r], it must return exactly one assertion for each SAML artifact found in the corresponding SAML request element [q of Step 4]. [SAML02b, p. 16]
- (g) For each SAML assertion included into r must hold that its AssertionID fulfills the constraint [CG1]:
- The [source site S] that assigns [the identifier AssertionID to the assertion] must ensure that there is a negligible probability that [the source site S] or any other party will accidentally assign the same identifier to a different data object.
 - Where the [SAML assertion] declares that it has a particular identifier [AssertionID], there must be exactly one such declaration.
- (h) [CP5h]: The source site [S] must implement an one-time request property for each SAML artifact. [SAML02b, p. 16]
- (i) [CP5i]: If a SAML artifact is presented to the source site again, the source site must return the same message as it would if it were queried with an unknown artifact. [SAML02b, p. 16]
- (j) [CP5j]: Artifact destination equality. The source site must return a response [r] with no assertions, if it receives a SAML request message [q] from an authenticated destination site \mathcal{X} containing an artifact issued by the source site to [the artifact receiver URL $\langle \text{AR-URL} \rangle_{\mathcal{Y}}$ of] some other destination site \mathcal{Y} , where $\mathcal{X} \langle \rangle \mathcal{Y}$. [$\mathcal{X} \langle \rangle \mathcal{Y} \Leftrightarrow id_{\mathcal{X}} \langle \rangle id_{\mathcal{Y}}$] [SAML02b, p. 16]
- (k) An honest destination site \mathcal{D} checks whether the $\langle \text{SR-URL} \rangle$ of source site S' it sent a SAML request q to in Step 4 is equal to the URL the corresponding response r comes from.

6. Response to \mathcal{B} :

- (a) If Step 4 or 5 were not successful or not all artifacts sent to S in Step 4:
 $\mathcal{D} \rightarrow \mathcal{B}$: Request of redirect to dereferer URL $\langle \text{DEREF} \rangle_{\mathcal{D}}$ of \mathcal{D} without SAML searchpart in the redirect.
 $\mathcal{B} \rightarrow \mathcal{D}$: Dereferer redirect
- (b) $\mathcal{D} \rightarrow \mathcal{B}$: Response with error code or target site.

We added a dereferer redirect to this Step 6 to prevent an adversary from seeing unused SAML artifacts in the HTTP Referer Tag after a failed SAML transaction between \mathcal{D} and \mathcal{S} .

The protocol we designed enhances the security of the SAML Single Sign-on profile by precautionary robustness measures and concrete extensions that anticipate the described attacks. It is downwardly compatible to the original SAML SSO profile. The measures we claim are realistic and can be provided by the most important protocol bindings.

Chapter 5

Conclusion and Future Work

In Part I we analyzed the SAML Single Sign-on Browser/Artifact profile. We refined this profile to a protocol description according to cryptographic templates which is suitable for a rigorous security analysis. We put together all assumptions and constraints of absolute requirement or prohibition in one place and extracted an explicit protocol schema. We provided the first exact analysis of a protocol standard such as SAML SSO and discovered multiple attacks on the SAML Single Sign-on Browser/Artifact profile. We proposed a repaired SAML SSO profile that is resistant against these attacks.

The refinement to a protocol schema with explicit assumptions and constraints eased the security analysis of the SAML SSO protocol. As all statements relevant for security of one protocol step are in one place, it is easier to spot weaknesses in the protocol standard. The attacks we described do not affect all protocol bindings or implementations of the SAML SSO profile. Especially the HTTP over SSL/TLS binding is resistant against different attacks, because it provides unilateral authentication where the profile does not claim it. Still, it should not be implementation- or binding-dependent, whether the concrete instance of SAML SSO is vulnerable or not.

Our results show that an exact analysis of protocol standards like the SAML Single Sign-on Browser/Artifact profile is possible. We think that a further formalization of the profile and its participants would be helpful.

We see that the exact and detailed formalization of a real-world concept like a web browser is still an open problem. It is non-trivial to formulate a polynomial-bounded probabilistic Turing Machine that covers all relevant aspects of a standard web browser. Nevertheless such a formalization would support analyses like that one we did for the SAML Single Sign-on Browser/Artifact profile.

In order to improve the security of protocol standards like SAML SSO it is crucial to develop formalization and proof techniques for standards and real-world protocols. The goal of our future work in this area is to provide watertight security proofs of these protocols directly based on the formulation of the standard. As first step in this direction, we try to prove the security of our improved SAML Single Sign-on Profile.

Part II

**Design of the
DynAdiEntitlementService**

Chapter 6

Introduction

Within this chapter we describe information related to our design project. On the one hand we introduce Access Control Products on the market as environment of our DynAdiEntitlementService. On the other hand we describe literature and design patterns related to our project.

6.1 Environment

Within this section we describe the environment in which context-based access control takes place. For a given access control solution we distinguish the involved products in two types important for our project: Access Control and Web-Access Enforcement Products.

An Access Control Product takes the access control decision for a given site. It possesses a security policy expressed in Access Control Lists (ACL), Capability Lists, Protected Object Policies (POP) or boolean rules. Given this policy it takes into account information about the user, his level of authentication, the context and the resource he wants to access to take an access control decision. The Access Control Product takes these decisions on request of another party. In our scenario a Web-Access Enforcement Product requests this decision and enforces it afterwards.

Such a Web-Access Enforcement Product is usually a proxy server on the boundary of a company's site. It filters the HTTP and HTTPS streams, authenticates users and maps the users' requests to certain resources. For each request the Web-Access Enforcement Product asks the Access Control Product for permission and provides the user data necessary for the decision. Receiving the result of the access control decision the Web-Access Enforcement Product opens a connection to the resource or responds with an error message.

In the following two sections we introduce the currently major products on the market. They all are capable of evaluation dynamic rules and therefore of handling dynamic attributes and do context-based access control. In Section 6.1.1 we describe the Access Control Products and in Section 6.1.2 the corresponding Web-Access Enforcement Products.

6.1.1 Access Control Products

In this section we describe the different competitors in the market of Access Control Products. We do not state a preference for one of these products and only want to

provide an overview over the market. As current market analysis are not public available, we refer to a competitive analysis of the Gartner group [All02]. As the market of these Access Control Products is immature and volatile, the order of market shares might have changed in meantime.

Netegrity Inc. SiteMinder SiteMinder is a directory-enabled Access Control Product that features dynamic content delivery, distributed administration, and integrated authentication for network access devices. According to a market share analysis of the Gartner Group [All02] Netegrity leads the market of extranet access management products. It has a larger number of production deployments than the Tivoli Access Manager and a growing number of alliances with major portal and e-business platform software vendors.

Tivoli Access Manager The Tivoli Access Manager (formerly known as Policy Director) is a policy-based single sign-on and authorization solution for e-business and legacy applications. This Access Control Product is designed to define and manage security policy for a broad range of e-business applications and other managed resources. According to [All02] the Tivoli Access Manager has the second-largest market share within the market of extranet access management products.

RSA Security Clear Trust As stated by [All02], ClearTrust is another significant challenger to the market leadership of Netegrity. ClearTrust offers both flexible and scalable access control services, including built-in support for delegated administration, and sophisticated auditing capabilities.

Novell iChain iChain is a directory-based Access Control Product featuring single sign-on, multi-factor authentication, and access control.

6.1.2 Web-Access Enforcement Products

The descriptions of the Web-Access Enforcement Products are also based on [All02].

Netegrity Inc. SiteMinder Secure Proxy Server The Secure Proxy Server of SiteMinder is deployed at the boundary of a companies site.

Tivoli WebSEAL WebSEAL is a high-performance web server that manages access control for different kinds of web-based resources. It is also used as proxy on the boundary of a company's site. It is designed to work together with the Tivoli Access Manager.

RSA Security ClearTrust Agent In spite of the other the Web-Access Enforcement Products the ClearTrust Agent is not a proxy server, but a plug-in for web servers and back-end servers and applications. It can be installed on a proxy server too, but is no proxy server.

Novell iChain Proxy Server As Netegrity and Tivoli Novell's iChain uses a proxy server to enforce access control decisions.

6.2 Related Literature

Within this section we describe literature related to our project.

We considered several publications about context-based access control and access control in general. The paper 'Access Control: Principles and Practice' [SS94] contains an

overview over static Access Control. It also describes the concept of the access matrix as well as access control and capability lists. A more theoretical introduction to access control can also be found in [Pfi00].

The high-level white-paper 'Differentiating Between Access Control Terms [Cam01] explains the difference between context-based access control and other types. One of the first implementations of context-dependent access control is described in [NHMT98] where the authors described a proprietary system for context-based access control in a hospital information system.

For the functionality of Access Control Products of of the most recent publications in 'The Authorization Service of Tivoli Policy Director' [Kar01]. This paper describes the functions and rough design of the Tivoli Access Manager (formally known as Policy Director).

As literature for the software engineering techniques that we apply and the waterfall model we follow, we used on the one hand 'Softwaretechnik I' [ZS02] and 'Lehrbuch der Software-Technik I' [Bal96] on the other hand. Both are lecture books written in German. You can also use Ghezzi's 'Fundamentals of Software Engineering' [GJM91]. A short review of the waterfall model itself can be found in [Kah01].

For the design, implementation and deployment of the Web Service components we used several publications. The book [Cer02] contains a general introduction to Web Service programming and WSDL files. We used the GLUE WebService Toolkit of Mind Electric¹ for the first prototypes and with it the GLUE user guide [Ele02]. Later we used the IBM WSTK Toolkit² and the corresponding introductions to Web Service architecture [Kre01] and implementation [Sne02].

6.3 Related Protocols

We distinguish two different types of protocols within this project. On the one hand fixed-provider protocols that deals with fixed servers and usually apply simple protocol flows. On the other hand we use browser-based protocols that involve a web browser and allow more complex flows. We described the browser-based protocols we utilize in Chapter 2. Within this section we describe the characteristics of both protocol class from a design point of view.

Within the implementation of the DynAdiEntitlementService we integrate a fixed-provider protocol into our framework. As the market of these protocols is subject of frequent changes³, we only describe usual characteristics of this type of protocols.

Protocol Flow: The fixed-provider protocols are in most cases Request-Response protocols. Normally the client sends a request to the server and that one responds with the desired attributes or an error message within the same communication channel.

Message Formats: Even though several proprietary message formats exist, we think there is a tendency towards the usage of XML messages. In most cases these messages are wrapped within a SOAP envelope. Some new protocols utilize the Security Assertion Markup Language (SAML) as message format for requests and responses. Another possibility is to use Web Service remote procedure calls to

¹<http://www.themindelectric.com/>

²<http://www.ibm.com/developerworks/webservices/wsdk/>

³The protocol we integrated in the DynAdiEntitlementService changed for instance three times within half a year.

retrieve attribute data. Our DynAdiEntitlementService provides this possibility as default interface.

Communication Bindings: Apart from proprietary implementations, the bindings most used are HTTP or SOAP over HTTP.

Security Measures: Normally the request is signed by the client and the response by the server. If the communication is not performed through a secure channel, the message is usually encrypted on message level. According to our observation it is more common to use HTTP over SSL 3.0 or TLS 1.0, which provides a secure communication channel. In most cases the channel utilizes at least unilateral authentication with a server certificate.

Considering the characteristics of browser-based protocols within the same scheme, we find the following properties:

Protocol Flow: Browser-based protocols usually involve three or more parties, whereas one of the participating parties is a web browser. This browser can be enhanced by a special protocol module (so-called enabled) or not. To transfer data with an unenabled browser these protocols usually use redirects with small pieces of information in the URL's searchpart or HTTP POST of HTML Forms.

Message Formats: Similar to the fixed-provider protocols there is a tendency to utilize XML messages for the messages send between non-browser parties. The browser-based protocols also use SOAP envelopes and the Security Assertion Markup Language (SAML) for these messages.

Communication Bindings: The communication that involves a browser usually uses a HTTP or HTTP over SSL/TLS binding. For other communication HTTP or SOAP over HTTP are likely bindings.

Security Measures: For the communication with browsers it is a common security measure to use HTTP over SSL or TLS. Since the browser has no own certificate, only unilateral authentication is possible in this case. For other communication the same considerations as for the fixed-provider protocols hold.

6.4 Design Patterns Used

One of the most important steps in evolution of software engineering was the development of generic design patterns. While architecture patterns model the relation between large components, the design patterns describe relationship and communication behavior of classes and objects. They provide generic solutions for common design problems and analyze the applicability for different situations.

One of the major publications in this area is the Erich Gamma's book Design Patterns [GHJV95], which is now widely-used in professional software design. Within the design part of this thesis we use several design patterns and rely partly on this book. We describe the five most important patterns in the following sections. In the sections 6.4.1 and 6.4.2 we introduce the creation patterns Singleton and Factory Method. Section 9.5.1 contains one of our own creation patterns: Single Instance per ID. Section 6.4.3 contains the structure pattern Adapter. In the sections 6.4.4 and 6.4.5 we describe the behavior patterns Strategy and State.

6.4.1 Singleton

The Singleton design pattern [GHJV95] only involves one class. It is one of the simplest patterns we deal with. We use it in situations in which we want to have an instance of a class globally accessible and guarantee that there is only one instance of this class. In the DynAdiEntitlementService we use the Singleton pattern for all classes providing data for the whole service.

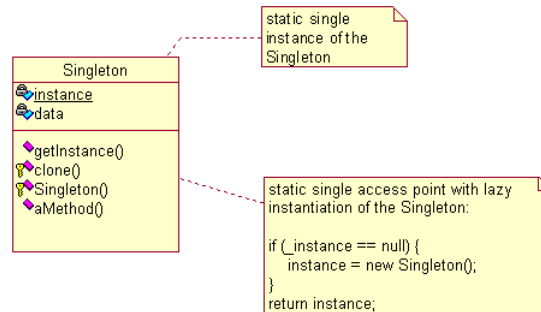


Figure 6.1: Class Diagram of a Singleton Class

As you can see in Figure 6.1, the `getInstance()` method of the pattern provides one static access point to the single instance of the class. We use this access point for a lazy initialization that guarantees that a call of `getInstance()` always returns an initialized instance of the Singleton. The constructor of the Singleton and the clone method explicitly prohibit external access such that the `getInstance()` is the only possibility to generate an instance.

6.4.2 Factory Method

The Factory Method [GHJV95] is a creation pattern with the purpose to dynamically change the created product at run-time. It's used to let subclasses of a Creator class decide which type of Product to create. We use this design pattern to provide extensibility for the types of DynAdiClients the service can produce and to delegate the creation of these clients to specialized factory classes.

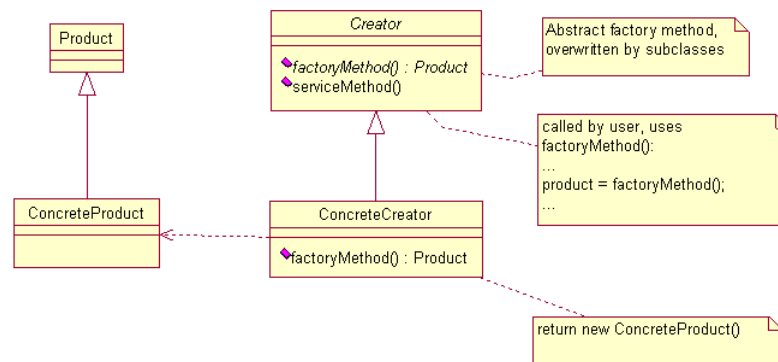


Figure 6.2: Class Diagram of the Factory Method Pattern

We depict the class diagram of this design pattern in Figure 6.2. The Creator class contains an (possible abstract) method factoryMethod() for the generation of the instances of Product. The concrete subclasses of Creator overwrite this method with one that generates instances of ConcreteProduct.

6.4.3 Adapter

Normally the Adapter design pattern [GHJV95] is used to adapt a given class to a new interface. Within the design of the DynAdiEntitlementService we use this pattern to anticipate future changes in the Web Service interface of our service. We adapt our main class to the stub files generated out of Web Service interface definitions in WSDL format. Therefore our service can support different Web Service front-ends.

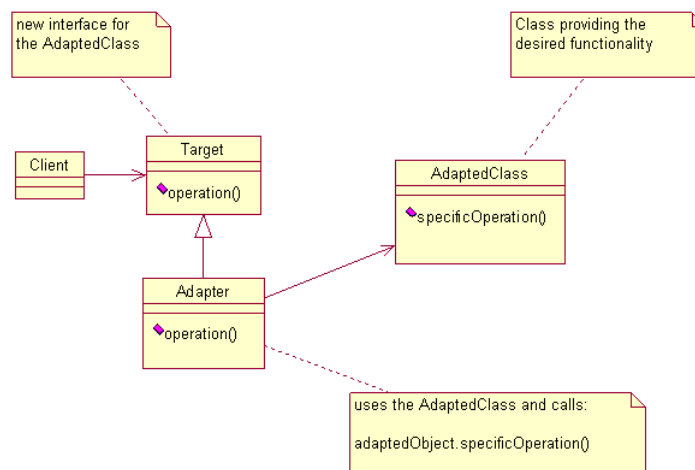


Figure 6.3: Class Diagram of the Adapter Design Pattern

In our scenario the Target class is the Web Service interface's stub class. The corresponding Adapter is a subclass of this Target class. It has a reference to an instance of the main class of our service (the AdaptedClass and delegates method calls to that one. The Adapter has to perform transformations from the data structures of the Web Service to that ones of our main class. We buy the higher flexibility with the additional expenses of this transformation.

6.4.4 Strategy

We also use the Strategy pattern [GHJV95] to allow more flexibility and anticipate future changes. It allows to modularize and exchange algorithms with the same interface. Where the Adapter pattern in section 6.4.3 allows changes of the interface to the Access Control Product, we use this design pattern to make the DynAdiProtocols exchangeable. A DynAdiProtocol is such to say a strategy of the DynAdiClient to communicate with the attribute provider.

The DynAdiClient is in the role of the Context, whereas the concrete variant of class DynAdiProtocol is a ConcreteStrategy. We use a variant of the Strategy pattern, whose Strategy is not stateless. The concrete instance of Strategy has a reference to the instance of its Context that allows the Strategy to access the whole interface of the Context.

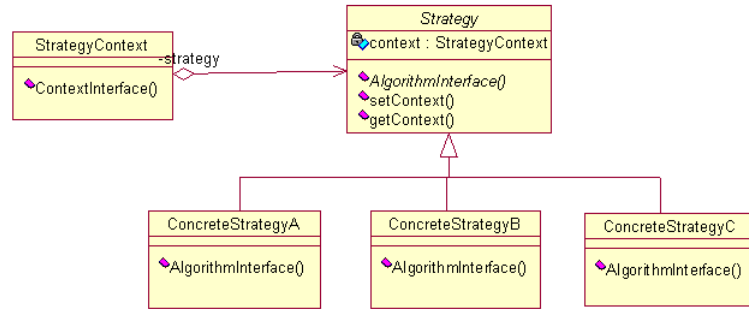


Figure 6.4: Class Diagram of the Strategy Design Pattern

6.4.5 State

The State pattern [GHJV95] is used to allow an object to change its behavior, when its internal state changes. Within this pattern you encapsulate a Context's state-dependent behavior in State objects. Within the design part of this thesis we use this pattern to model complex protocols with multiple states.

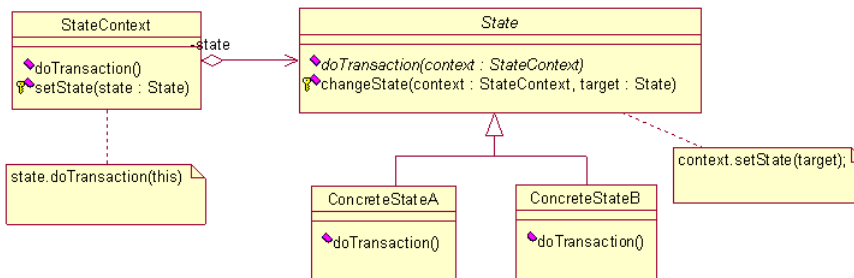


Figure 6.5: Class Diagram of the State Design Pattern

We depict the class structure of the pattern in Figure 6.5. The Context class delegates state-dependent method calls to the current State instance and hands over a reference to itself. With this reference the concrete instance of State can perform the operation with access of the context instance's state. The instance of State not necessarily needs to be stateful.

Chapter 7

Design Overview

In this chapter we present an overview of the design for the `DynAdiEntitlementService`. It serves as introduction for the following chapters that address the detailed requirements of the project, its high-level design and implementation issues.

We designed and implemented the `DynAdiEntitlementService` following the so-called waterfall model that is described in [ZS02] and [Bal96]. This model is a traditional model and quite intuitive. It organizes the software engineering process in strictly ordered phases. The work for one phase can only start, if the previous phase was finished successfully. We depict the phases of the model in Figure 7.1.

The waterfall model makes two critical assumptions: On the one hand it assumes that the project's requirements and specifications do not change within the project. If this happens, one normally has to go back phase by phase to cope with the changes, which is very expensive. Thus, the utilization of this process model implies a high priority of anticipation of future change. On the other hand the model assumes that the software engineer is able to understand all aspects of each phase. Therefore the model requires a careful analysis in each phase.

This chapter itself reflects the feasibility phase of our project and describes the major challenges and use cases. The requirement phase of the model is described in chapter 8 (Page 75). It contains the functions and goals we expect to fulfill with our product. We describe the high-level level design of the `DynAdiEntitlementService` and analyze chosen problems of the low-level design in Chapter 9 (Page 76). The high-level design organizes the product in classes and describes their relationship. The low-level design specifies the behavior of the classes on method-level and contains detailed collaboration descriptions. The Chapter 10 (Page 95) is concerned about the implementation phase as well as issues from the testing and deployment phases.

We describe the components we deal with in Section 7.1. In Section 7.2 introduce the design challenges we identified during the first analysis of this project. In Section 7.4 we depict the general high-level protocol flow of the `DynAdiEntitlementService` that solves the main goal of the project. We show the most important use cases of the project in Section 7.5.

7.1 Components that Interact

We distinguish the following components and depict their relation in Figure 7.2. The components belonging to the DynADI project are colored dark.

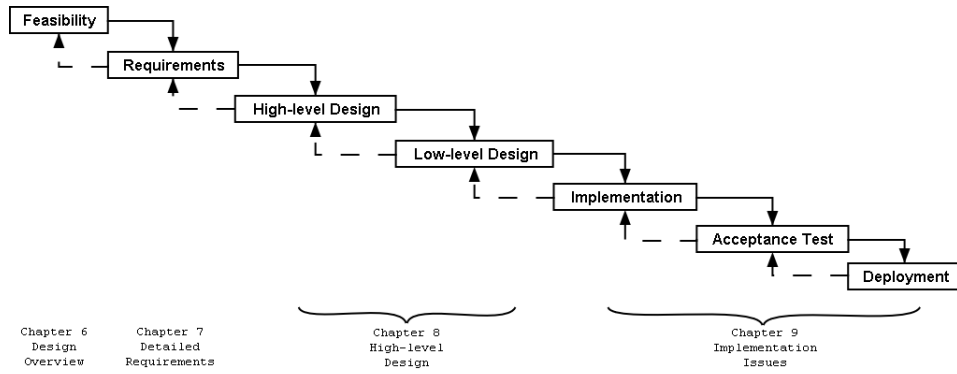


Figure 7.1: Utilization of the Waterfall Model as Process Model

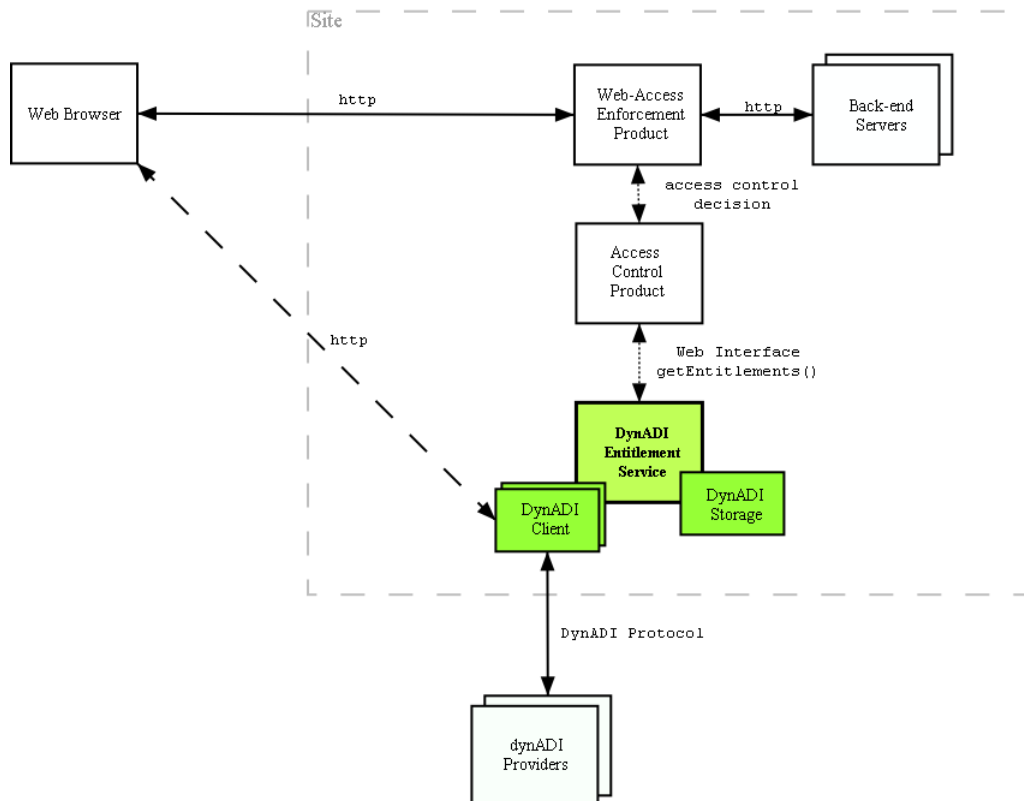


Figure 7.2: Components that Interact

Web Browser A browser that tries to access web pages on one or more back-end servers secured by an access control product.

Back-end Servers These are servers with sensitive or valuable data. A Web-Access Control Product secures them.

Web-Access Enforcement Product This system shields and protects resources on the back-end servers. It asks an access control software for authorization. The system enforces the decisions taken by the access control product.

Access Control Product managing access control and deciding authorization queries. It has the capability to evaluate boolean rules on the Access Decision Information (ADI).

DynAdiProvider A service that can provide ADI. Such a provider can be web-based or not. It may be on a local web server or not.

DynAdiProtocol A particular entitlement protocol of a given vendor. Such a protocol is used to communicate with a DynAdiProvider.

DynAdiEntitlementService is used to retrieve missing ADI on request. It encapsulates different entitlement provision protocols. This is the central component of our project. For short we also call this service *DynADI Plug-in*. The DynAdiEntitlementService is associated with two other components: the *DynAdiStorage* and the *DynAdiClient*, which are described below.

DynAdiStorage This component caches the DynADI temporarily. The duration is specified by the protocol used. The DynADI may also be stored in the state of a *DynAdiClient*.

DynAdiClient A client module of the DynAdiEntitlementService that runs a special DynADI Protocol in order to communicate with one DynAdiProvider.

7.2 Design Challenges

In this section we describe the environment and design challenges we have to deal with.

7.2.1 Product Environment

The DynAdiEntitlementService will be included in a given Access Control Product written in the programming language C. Our service is supposed to be written in Java to interface with the DynADI Providers' libraries that are often provided in Java. The current architecture of the Access Control Product is not designed, but grown over several years. Its infrastructure originally was not designed for the support of dynamic attribute provision. Therefore the given interfaces do not satisfy our needs.

We are required to integrate our service to the given software and cannot develop its design from scratch. Therefore the environment of our product is a main constraint for possible solutions. Considered the facts above we classify the product environment as non-optimal.

7.2.2 Quality Requirements

Given the product environment in Section 7.2.1 we have to fulfill several quality requirements. The DynAdiEntitlementService is directly included in the Access Control Product. It has to be product-quality with our shipment and fulfill the security and reliability requirements of professional access control products. Therefore it needs to be highly robust and provide a good performance and through-put. It has to be maintainable by developers not directly involved in the product design and implementation. Therefore we need to keep the design as intuitive and understandable as possible.

As the dynamic entitlement provision is an emerging technology, it is suspect to frequent change. We need to anticipate future changes in interfaces and protocol flows of entitlement protocols as well as in the interface to the Access Control Software. Our product has to be very modular and extensible.

7.2.3 Resulting Challenges for our Design

We were required to provide a above-average product quality in a difficult product environment. In this section we name concrete design challenges we faced in the development of the DynAdiEntitlementService.

Robustness and Testing The robustness requested by the customers implies an at least semi-formal specification method and excessive testing. As provable correctness is beyond the scope of this thesis, we have chosen exemplary specification with test cases as method and use junit-based automated class tests as well as an external test harness. We describe this technique in Section 10.1.1 (Page 95).

Interfacing Java and C Given that we had to bind together Java-based protocol libraries and an Access Control software in C, we have to choose a appropriate position of a bridge and a technology to build this bridge. We address this problem in Section 9.2.3 (Page 80).

Minimal changes to the Access Control Product Our project should only expect minimal changes from Access Control and Web-Access Enforcement Products. Therefore we have to cope with given interfaces that are not designed for provision of sufficient information for dynamic attribute retrieval. The flow of Section 7.4 allows the retrieval with only marginal changes to the given products.

Obtaining Browser Focus Especially for DynADI Protocols that need to communicate with the user's browser, we need techniques to obtain the browser focus from the Web-Access Enforcement Product. Unfortunately the given product has no functionality to hand over the browser focus to another party. We solve this problem with the flow of control of Section 7.4.

Generic Protocol Interfaces There are many different proprietary protocols for attribute retrieval. These protocols need different information to work and provide the retrieved attributes in different formats. We cope these possibilities of change in two ways. On the one hand we provide high modularity for the different protocol modules and very generic interfaces. On the other hand we use a generic data type called Container to transport attribute data. The format of a Container's payload is not fixed, but protocol dependent and described in so-called ContainerDescriptors. We describe the relation of both classes in Section 9.2.2 (Page 78).

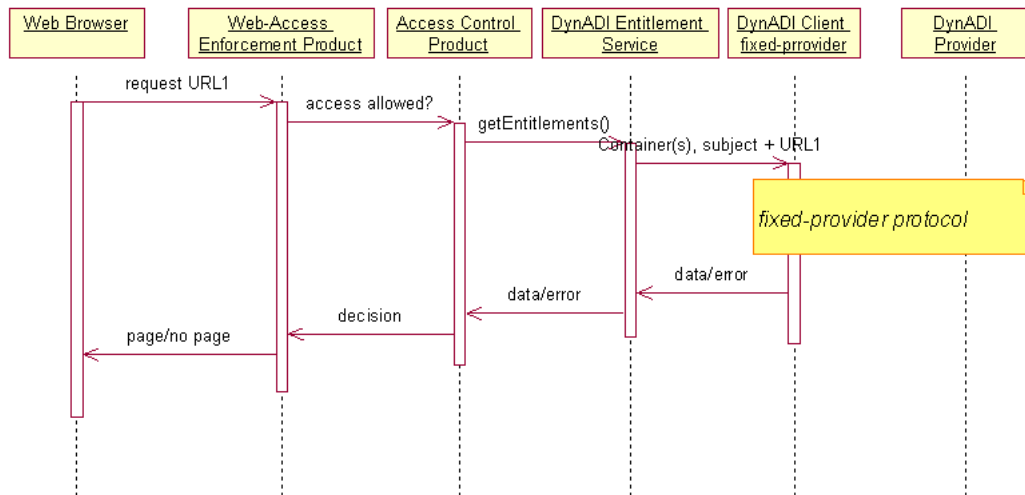


Figure 7.3: Sequence diagram: Integrating a Fixed-provider Entitlements Service

7.3 High-level Flow for Fixed Providers

In this section we describe the protocol flow an attribute retrieval with fixed providers. This flow is similar to the first part of the generic flow in Section 7.4.

In Figure 7.3, we depict the simplified flow for a particular fixed-provider entitlements service. Under the assumption that retrieval of entitlements is faster than the timeout of the HTTP request, the query can be implemented in phase 1 of the ADI protocol.

In case the query time exceeds the timeout, phase 2 of the DynADI protocol can simply be a Web page asking to wait until the collection of required entitlements has finished.

In the following paragraph we describe the steps in detail:

1. Web browser sends a request for a resource under URL1.
2. Web-Access Enforcement Product intercepts the request and asks the corresponding Access Control Product for authorization.
3. Access Control Product evaluates the security policy and discovers a rule associated with the resource (e.g. *user_age* >= 18).
4. The Access Control Product determines the required dynamic attributes from the rule (e.g. *user_age*). The Access Control Product calls the DynAdiEntitlementService using a Web Service client.
Inputs: type of missing attributes, target URL1 and information about the user.
5. The DynAdiEntitlementService returns the required entitlements, if they are already cached (flow description not continued). Otherwise, the DynAdiEntitlementService determines appropriate protocols and DynADI Clients for obtaining the different kind of containers.
6. **DynADI protocol phase 1:** The DynAdiClient connects to the corresponding attribute provider. It directly retrieves the required information and returns it. The DynADI client may, but need not cache the received ADI.

7. The DynAdiEntitlementService returns the entitlements from the DynADI Client to the Access Control Product.
8. Access Control Product uses the returned DynADI to evaluate the rule attached to the accessed resource. It returns its decision to the Web-Access Enforcement Product.
9. The Web-Access Enforcement Product allows or denies access to URL1 according to the Access Control Product's decision.

7.4 Generic High-level Flow

In this section we describe a generic high-level flow that solves integrates the flows for fixed-provider and browser-based protocols.

In Figure 7.4, we illustrate the high-level flow of control among the interacting components. We considered several flow options, of which this one seems the best choice. It also requires only minimal changes to a Web-Access Enforcement and Access Control Product.

The basic idea of the flow is to use redirects to obtain the browser focus. The DynAdiEntitlementService sends a redirect request back to the Web-Access Enforcement Product. The browser then redirects to a source site or the DynAdiEntitlementService itself. The browser then participates in a browser-based protocol that fulfills the following post conditions. On the one hand it hands all requested attributes to the DynAdiEntitlementService and redirects the browser back to the originally requested URL.

In the following paragraph describe the flow in detail:

1. Web browser sends a request for a resource under URL1.
2. Web-Access Enforcement Product intercepts the request and asks the corresponding Access Control Product for authorization.
3. Access Control Product evaluates the security policy and discovers a rule associated with the resource.
4. The Access Control Product determines the required dynamic attributes from the rule. The Access Control Product calls the DynAdiEntitlementService using a Web Service client.
Inputs: type_ids of missing attribute containers, target URL1 and information about the user.
5. The DynAdiEntitlementService returns the required entitlements, if they are already cached (flow description not continued). Otherwise, the DynAdiEntitlementService determines appropriate protocols and DynADI Clients for obtaining the different kind of containers.
6. **DynADI protocol phase 1:** If possible, the DynAdiClient directly retrieves the required information and returns it. We do not describe the canonical flow of the information back through the DynAdiEntitlementService, Access Control and Web-Access Enforcement Product here. The DynADI client may, but need not cache the received ADI.
Otherwise, the DynADI Client returns a redirect URL possibly including a fresh session-ID. This session-ID later enables the DynADI Client to link the incoming HTTP request (from the redirect) with the target URL and the requested data.

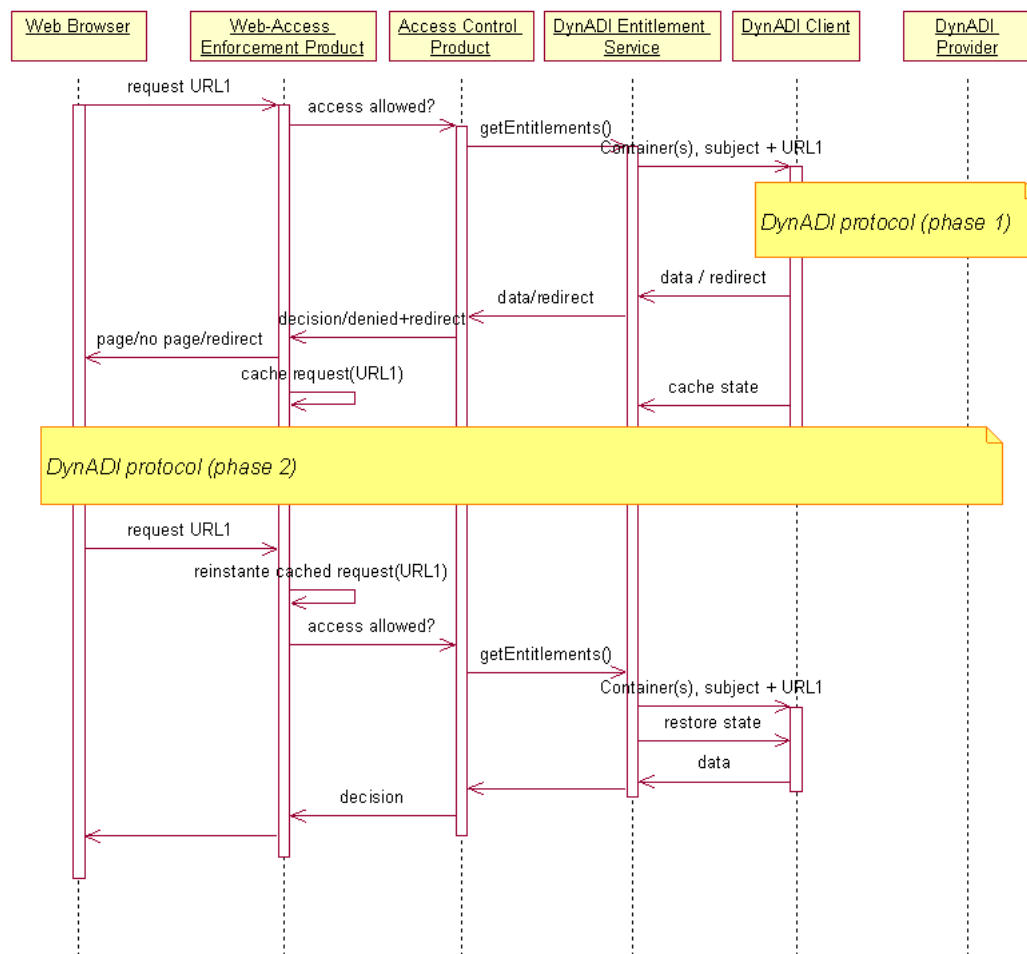


Figure 7.4: Sequence diagram: Integrating Dynamic ADI Retrieval

7. The DynAdiEntitlementService returns the redirectURL from the DynADI Client to the Access Control Product.
8. Access Control Product returns a deny decision and the redirectURL to the Web-Access Enforcement Product.
9. The Web-Access Enforcement Product denies access to URL1 while redirecting the browser to the given URL (typically of the DynADI Client, possibly also the DynADI Provider) within the HTTP response.
10. **DynADI protocol phase 2:** If the Web browser was redirected to the DynADI Client¹, it executes the (federated) DynADI retrieval protocol with the browser and the DynAdiProvider. It knows by the session-ID (carried in the redirectURL) which containers to ask for. If the DynADI Client redirected directly to the DynADI Provider in DynADI protocol phase 1, it is protocol-dependent how the DynADI Client is next contacted and where the session ID is encoded. The protocol must end such that the browser is redirected back to its target URL1, possibly with an additional element in the search string of URL1, e.g., the session-ID again. The DynADI Client must cache the retrieved ADI.
Note: The given Web-Access Control Product caches incoming requests of authenticated users before redirecting the browser within a DynADI protocol run. It reinstantiates the cached request when redirecting back to URL1. Therefore it is necessary to redirect back to the unmodified URL1.
11. Web browser sends a request to URL1 (automatically, triggered by the last redirect of the DynADI provider).
12. The Web-Access Enforcement Product intercepts request and asks the Access Control Product for authorization.
13. The Access Control Product evaluates the security policy and discovers a rule associated the the resource URL1.
14. The Access Control Product detects that some ADI is missing for being able to evaluate the rule. It calls the DynAdiEntitlementService.
15. The DynAdiEntitlementService either directly retrieves the entitlements from its cache or it determines and calls the appropriate DynADI Client for obtaining this container type as above.
16. The DynAdiEntitlementService returns the required data.
17. The Access Control Product evaluates the rule and returns decision.
18. WebSEAL allows/denies request.

7.5 Use Cases

A Use Case defines the intended behavior of a system when an actor (i.e. a user or an interacting component) sends a particular stimulus. It also specifies the outputs and reactions the actor receives from the system.

Given the generic flow of Section 7.4 as motivation, we discuss several use cases for this scenario in this section. We consider the Access Control Product and attribute providers

¹For protecting against denial of service attacks, the policy may sometimes restrict access to the DynAdiClients to authenticated users.

as actors and our entitlement service as system they deal with. We do not describe all use cases and the presented ones not in full detail. The purpose of these use cases is to get a feeling for the behavior of our service.

Use Case 1: Attribute retrieval from fixed providers: We depict this use case in Figure 7.5. Precondition: The DynAdiEntitlementService is initialized.

1. The Access Control Product sends a getEntitlements() request to the service.
2. It supplies sufficient subject information and the description of the attributes to be retrieved in the request.
3. The service performs input checks on all data of the request.
4. The service checks the DynAdiStorage for cached attributes.
5. The DynAdiEntitlementService finds out suitable protocols and providers.
6. It connects to the provider's servers.
7. It uses the entitlement protocols to retrieve the attribute data.
8. The entitlement service returns the attribute data and an error code for a successful protocol run to the Access Control Product

Error Case 1: Attributes could not be retrieved

Because of communication failures or the unavailability of the requested data at the provider's server the service cannot retrieve all attributes.

In Step 8 the service returns an appropriate error code for this case. It includes a detailed error description in the return as well as all attribute data it could retrieve in spite of the error. If the service is configured to use a certain browser-based protocol in the case it could not retrieve all ADI, it additionally returns the corresponding redirect URL.

Error Case 2: Request with malformed inputs

The Step 2 getEntitlements() request of the Access Control Product contains insufficient or malformed data. After the input checks of Step 3 it returns the corresponding error code and provides a detailed description of the incorrect inputs.

Error Case 3: Request to an uninitialized service

The Access Control Product sends the Step 1 getEntitlements() request to an uninitialized DynAdiEntitlementService. The service directly returns the corresponding error code and a detailed error description.

Use Case 2: Attribute retrieval of already cached data: This use case is based on 'Attribute retrieval from fixed providers'.

1. Replace Step 4 of use case 'Attribute retrieval from fixed providers' with:
The service checks the DynAdiStorage for cached attributes. All requested attribute data is cached within the DynAdiStorage.
2. The DynAdiEntitlementService retrieves the data from the cache.
3. The service returns them directly to the Access Control Product.

Use Case 3: Attribute retrieval with a browser-based protocol: This use case is based on 'Attribute retrieval from fixed providers'. We depict it in Figure 7.6

1. Replace Step 5 of use case 'Attribute retrieval from fixed providers' with: The DynAdiEntitlementService finds out the appropriate protocols and providers. The request implies an attribute retrieval with a browser-based protocol.

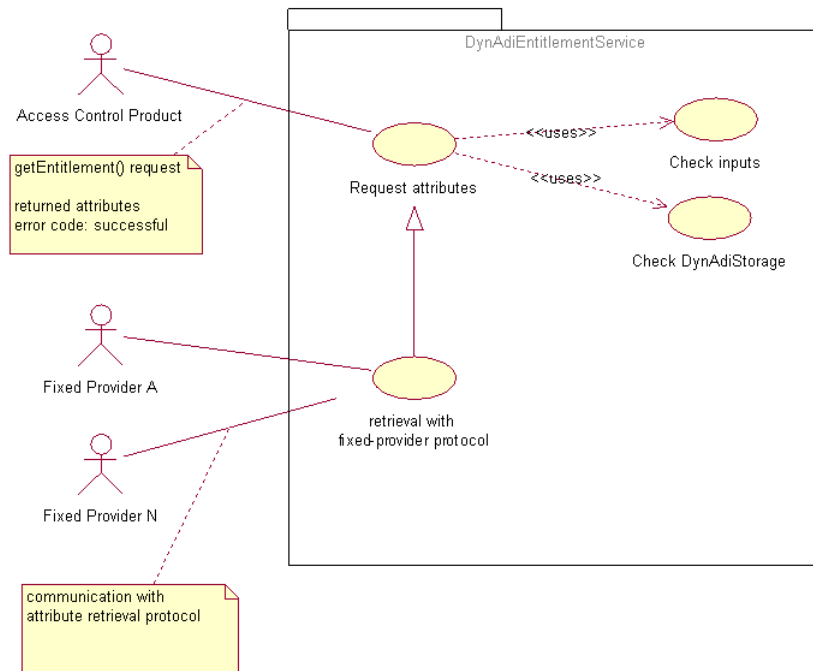


Figure 7.5: Attribute Retrieval from Fixed Providers

2. The `DynAdiEntitlementService` runs the corresponding protocol module.
3. It returns on its command all attributes already available, the redirect as well as a corresponding error code.
4. The `Access Control Product` responds with the given redirect to the user's browser.
5. The browser redirects to the `Source Site` given in the redirect (or to the `DynAdiEntitlementService`).
6. `Source Site` and browser communicate to allow the entitlement.
7. The `Source Site` communicates with the `DynAdiEntitlementService` to transfer the attributes.
8. The `Source Site` redirects the browser back to the `Access Control Product`.
9. This use case uses 'Attribute retrieval of already cached data' to respond to this request.

Use Case 4: Initialization request: Precondition: The `DynAdiEntitlementService` is not yet initialized.

1. The `Access Control Product` sends an `initialize()` request to our service.
2. The `DynAdiEntitlementService` loads the configuration, the key store and the data tables.
3. It performs a consistency check on the loaded data and does all pre-computation possible at initialization time.
4. If the initialization was successful, it returns a corresponding error code to the `Access Control Product`.

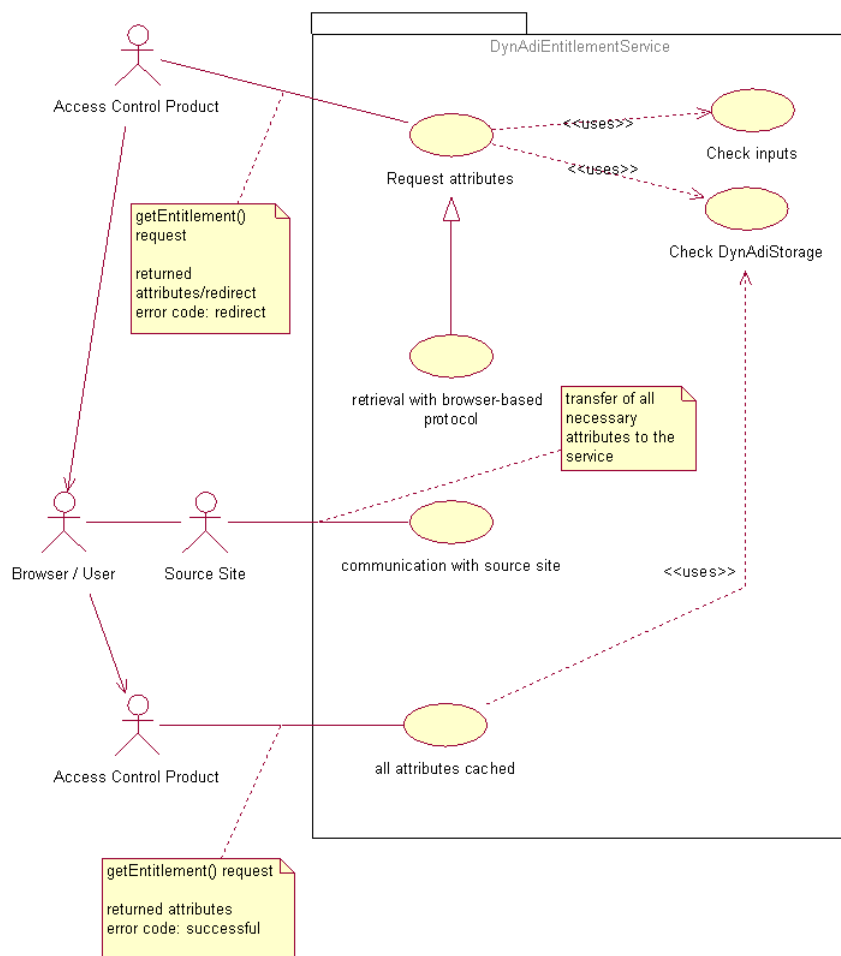


Figure 7.6: Attribute Retrieval with a Browser-based Protocol

Error Case 1: IO Exception

if the Step 2 IO operations fail or the data tables cannot be parsed, the service returns an initialization error and a description of the error cause.

Error case 2: Inconsistent data tables:

If the Step 3 consistency check notices an anomaly in the data tables or the relationship of the different tables, it removes the elements involved in the inconsistency. It returns an initialization error and a detailed description of the elements that cause the inconsistency.

Error Case 3: Second initialization

If Access Control Product sends a Step 1 initialize() request to an initialized DynAdiEntitlementService, the service does not perform a second initialization. Instead it returns an error code corresponding to a successful initialization and a error message describing the second call.

Chapter 8

Detailed Requirements

In this chapter we shortly introduce the requirements that our project needs to fulfill. The detailed requirements given in this chapter refine the rough requirements that we state in the design overview in Chapter 7. According to the waterfall model we here specify a product that is able to accomplish the use cases and protocol flows given there. We explicitly describe functions of the product, which helps to identify class hierarchies in the high-level design.

We formalize the detailed requirements according to the schemes of [ZS02] and [Bal96]. This chapter itself only contains a short informal introduction. We describe the formal requirements in detail in the Appendix Chapter A.

Goal The most important goal of our project is to provide a reliable and robust service which retrieves dynADI. It is supposed to communicate with fixed providers. We integrate this service into an existing Access Control Product and implement a flow of control that requires only minimal changes of the Access Control Product.

Quality Requirements We expect reliability as most important quality requirement. This means that the service behaves according this specification in most cases. This implies the usage of semi-formal specification and exhaustive testing.

The service has to be modular. The protocol modules used have to exchangeable easily.

We want the service to be robust. It is supposed to check all input values from the Access Control Product as well as from the attribute providers. It shall handle exceptions, errors and inconsistent states with care, report them to the Access Control Product and always get back to a consistent state.

Chapter 9

High-level Design

Our goal is to improve a given access control software by allowing the dynamic retrieval of additional Access Decision Information (ADI) when it is needed. We design a plug-in for the such a software product that performs the retrieval of these dynamic ADIs (for short we call them DynADI). The plug-in is called DynADI Entitlement Service and implements an entitlement provider interface of the access control software.

This chapter describes the high-level design of the DynAdiEntitlementService. The requirements are outlined in Chapter 8. Note: In this chapter we refer to function and test case numbers that are defined in Appendix Chapter A. Their format is [{RF,RT,RD}number] where RF stands for a function, RT for a test case and RD for product data. For instance [RF100] refers to the function 'Communicate with fixed provider' as described on page 111.

9.1 High-level Architecture

In this section we provide an overview over the architecture and high-level design of the DynAdiEntitlementService from a bird's view (see Figure 9.1).

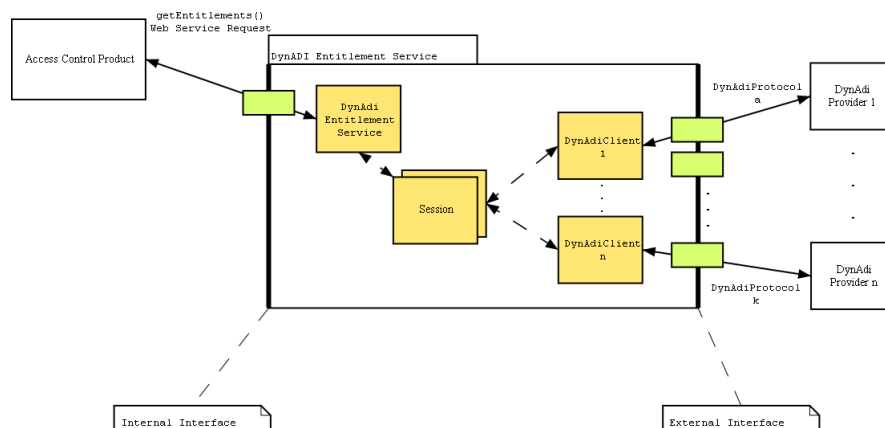


Figure 9.1: High-level Architecture of the DynAdiEntitlementService

The DynAdiEntitlementService communicates with two other kinds of parties. On the one hand there is an Access Control Product. The DynAdiEntitlementService

implements a given interface providing entitlements to this entity. Within Figure 9.1 we call this interface internal. We describe it in Section 9.2 (Page 77).

On the other hand there are the different DynAdiProviders that provide the dynamic attributes. The DynAdiEntitlementService uses special DynAdiProtocols to communicate with them. We call these interfaces external and describe them in Section 9.3 (Page 82).

The DynAdiEntitlementService has several core components:

DynAdiEntitlementService: provides the interface to the access control product.

It is responsible for the implementation of the given entitlement provider interface and its provisioning as a Webservice. For each getEntitlements() the DynAdiEntitlementService generates a new Session to retrieve the requested DynAdiContainers. Note that the DynAdiEntitlementService has the capability to manage several Sessions simultaneous. We describe this component in Section 9.4.1 (Page 83).

Session: manages the retrieval of the different DynAdiContainers requested by the access control product. It determines the order of the retrievals and combinations of provider and protocol used to get the DynAdiContainers. The Session generates a DynAdiClient for each combination of DynAdiProvider and DynAdiProtocol used to retrieve DynAdiContainers. In Section 9.4.3 we introduce this component and the related SessionTable.

DynAdiClient: This is the component that performs the retrieval of the DynAdiContainers. It contains a protocol-specific component that communicates with one certain DynAdiProvider. In a setting with fixed provider it connects to the provider's server and runs the DynAdiProtocol with it. We describe the class DynAdiClient in Section 9.4.4.

DynAdiProtocol: An instance of this class encapsulates a protocol engine for a specific entitlement provision protocol and its internal state. It has the capability to connect to a provider and to retrieve data from it. We describe the class DynAdiProtocol in Section 9.4.5.

DynAdiProvider: An object of DynAdiProvider contains the data of a certain server of an attribute provider.

9.2 Internal Interface

Within this section we describe the design of the interface to the Access Control Software.

9.2.1 Abstract Interface

In this section we identify the abstract interface to the access control software. We describe the inputs necessary for the DynAdiEntitlementService and the outputs required by an access control software.

Input

Subject: The `DynAdiEntitlementService` needs information about the subject of a request. On the one hand we need a unique identifier for a subject in order to distinguish different subject's non-ambiguously and to link DynADI for the same subject together. On the other hand we need subject attributes the `DynAdiProviders` can use to identify the subject. We consider the information attribute providers require in Section 9.3.2 on Page 82.

Requested attributes: The access control software has to request specific DynADI from the `DynAdiEntitlementService`. We assume that the entitlement request explicitly names the attributes to be retrieved.

Application specific information: To design the interface extensible we use another input value that allows to hand over further information about the handling of the `getEntitlements()` request. This value for instance may contain information about the dependencies of the different DynADI or the rule the access control software evaluates. It may contain restrictions about the usable protocols and providers or the caching policy.

Output

Retrieved attributes: The `DynAdiEntitlementService` returns all requested attributes it was able to retrieve to the access control software. In the case of an error the service returns the attributes retrieved so far. We describe the `DynAdiContainers` used to transfer the attributes in Section 9.2.2.

Target of a redirect: To enable browser-based protocol flows it is necessary that the access control product redirects the subject's browser to URLs the `DynAdiEntitlementService` specifies.

Application specific information: To anticipate future changes we provide a general output value that allows to return further application specific information to the access control software. Within this value the `DynAdiEntitlementService` could specify for instance details about the handling of the subject's HTTP session by the access control software.

Error descriptions: It is possible that multiple errors occur within a single `getEntitlements()` request, because the `DynAdiEntitlementService` tries to retrieve as much attributes as possible. Therefore we need an interface part that allows to return several error descriptions in a human readable format.

Major error code: Additionally to the error descriptions we return an integer error code, that specifies the most important major error according to the specification of the access control software.

9.2.2 DynAdiContainers that Abstract from Data Formats

The `DynAdiEntitlementService` provides the attribute data in so-called containers. Such a attribute container can hold virtually any kind of data. Usually its data is expressed in XML in a format known by the Access Control Product. We use this concept as abstraction from the concrete provider's format.

The instances of `DynAdiContainer` will store the information retrieved by the `DynAdiEntitlementService`. How the attributes are stored in the containers is defined by so-called container descriptors. Each container type is related to a special container

descriptor. Each container descriptor has unique container_type_id. A container described by a special descriptor has the same container_type_id.

The container itself contains the container_type_id and the retrieved attributes as payload. The payload is a XML Document that validates under the payload format defined by the container descriptor.

We describe the relationship between the container and container descriptor in Figure 9.2.

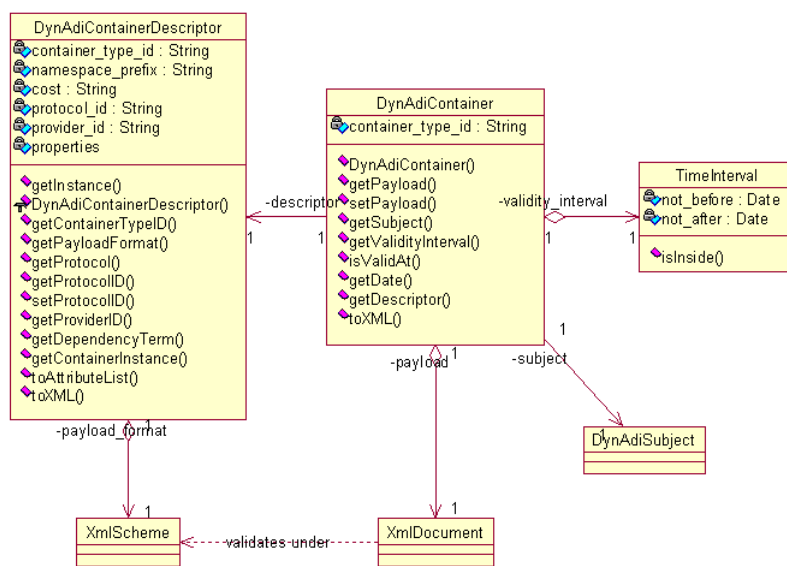


Figure 9.2: Class Diagram of Container and Container Descriptor

The XML structure that is defined by a DynAdiContainerDescriptor, i.e., an actual simple container, looks as follows:

```

<container_type_id>
  <attributeName1>data1</attributeName1>
  <attributeName1>data2</attributeName2>
  ...
  <attributeNameN>dataN</attributeNameN>
</container_type_id>
  
```

The DynAdiEntitlementService stores all container types it supports in a XML database. Only XML data defined by these container types can be used for building rules.

As a consequence, each rule contains a list of container_type_ids that are needed to evaluate this particular rule. Before evaluating the rule, the access control software asks the DynAdiEntitlementService to retrieve the required set of containers.

9.2.3 DynAdiEntitlementService as WebService

We utilize the WebService technology to connect the Access Control Product and the DynAdiEntitlementService in spite of boundaries of programming languages. We wrap the interface of the DynAdiEntitlementService class in a WebService. The team of the Access control Product implements a C WebService client for our service.

We build a bridge between both products as follows:

1. The team of the Access Control Product designs a C interface the Access Control Product is supposed to communicate with.
2. Out of the corresponding C header file they generate a description of the corresponding WebService interface. It is formulated in the Web Service Description Language (WSDL).
3. Given this WSDL file, we generate a set of Java stub files that implement the desired WebService interface.
4. We transform the data-structures of the WebService interface to appropriate Java data-structures and use them for calls to the DynAdiEntitlementService class. We describe the corresponding design in Section 9.4.2. We publish the WSDL file as description of our service.

With this integration done we have a WebService client on the side of the Access Control Product and a WebService with WSDL description on the side of the DynAdiEntitlementService. Our entitlement service is supposed to be deployed not only on application servers specialized in WebServices (i.e. the GLUE toolkit) but also on Servlet Application Servers (i.e. Apache Tomcat or IBM WebSphere AS). Therefore we integrate the our WebService in a Apache AXIS Servlet container using the IBM Web Services Toolkit. The container provides all infrastructure necessary to host WebServices utilizing several Servlet classes.

Putting this all together we have the design we depict in Figure 9.3. As you can see in this Figure Access Control Product and DynAdiEntitlementService use the WebService infrastructure to transform requests and responses into SOAP messages. Using this technique the communication between both products works as follows. We depict the flow as collaboration diagram in Figure 9.4.

1. The Access Control Product calls `get_entitlement()` function of its C API. This call is delegated to the corresponding WebService client.
2. The WebService client and the underlying C SOAP engine transform the function call to a SOAP request corresponding to the WSDL file of the WebService. This request is sent through a secure channel to the DynAdiEntitlementService WebService endpoint.
3. The Java SOAP engine of the WebService' Servlet container transforms the SOAP request to a Java `getEntitlements()` method call of the WebService Adapter of the DynAdiEntitlementService.
4. After performed the necessary computations and attribute retrieval the Java method returns providing the retrieved values.
5. The Java SOAP engine transforms this return into a SOAP response and responds with that one within the same secure channel to the C WebService Client.
6. The WebService client's `get_entitlement()` function returns with the values provided in the SOAP response.

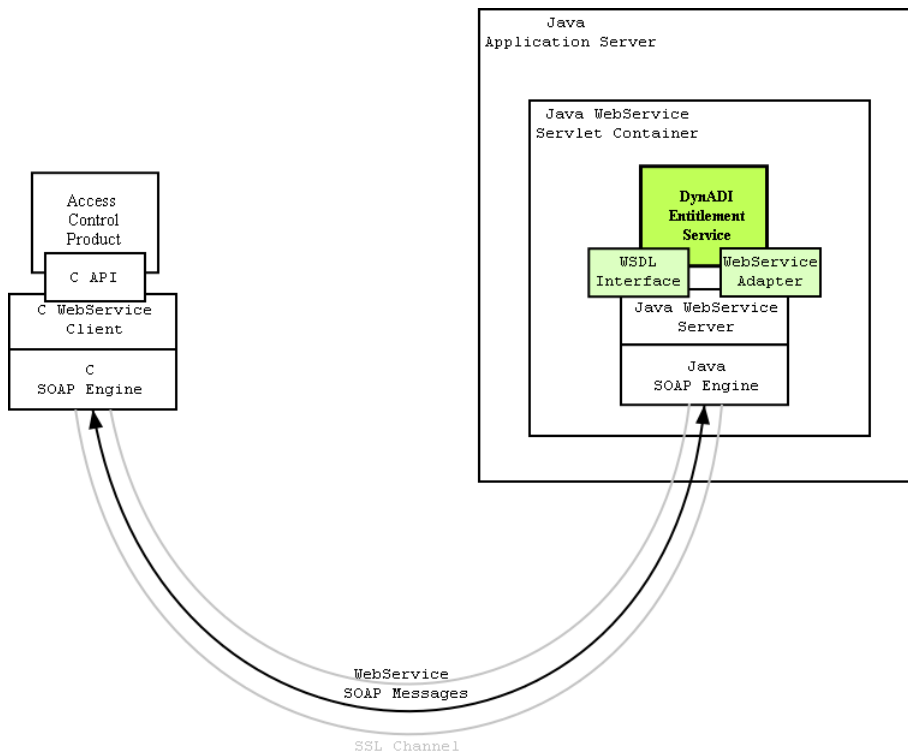


Figure 9.3: Connecting Service and Access Control Product with WebServices

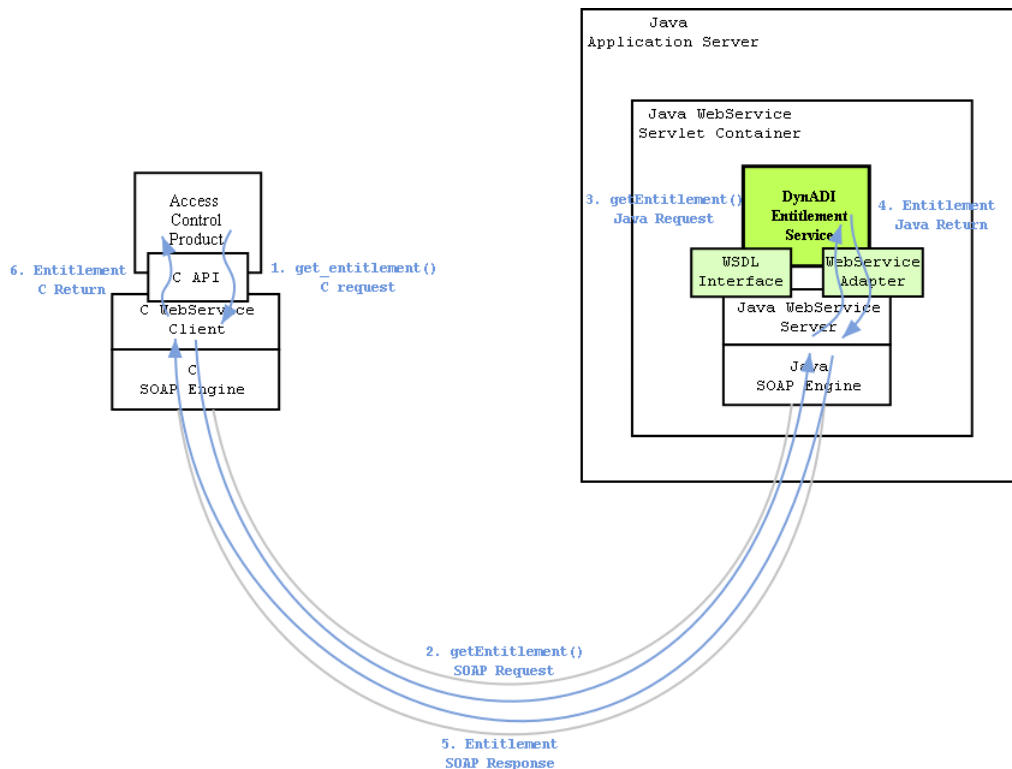


Figure 9.4: Collaboration Diagram with the WebService Infrastructure

9.3 External Interface

The `DynAdiEntitlementService` communicates with different `DynAdiProviders` and therefore uses several `DynAdiProtocols` which can be thought as external interfaces. In the first phase of the project the service will only communicate with one certain attribute provider using a proprietary implementation of the Security Assertion Markup Language (SAML) [SAML02a].

9.3.1 General Considerations

As far as we can see, within the project the interface to the attribute providers has the highest probability to change. Many `DynAdiProviders` use proprietary `DynAdiProtocols` to transfer DynADI. There are several common protocols that perform this transfer and might be subject to change.¹ Therefore it is important to keep the interface to the attribute providers most general and anticipate all changes that might occur.

Because of this anticipation of change we have to consider:

Separation of provider and protocol: We distinguish `DynAdiProvider` and `DynAdiProtocol` classes. While the `DynAdiProvider` bundles all information about a certain attribute provider and its server, the `DynAdiProtocol` manages the protocol flow and communication with the attribute provider. We require a strict separation of interests between these classes. We describe the class `DynAdiProtocol` in Section 9.4.5.

Generic class interfaces: Especially the interface of the class `DynAdiProtocol` has to be absolutely general. This allows to describe very different protocols with the same `DynAdiProtocol` interface.

Dynamical load of classes: We want to provide an easy extensibility. Therefore the protocols and providers that are available have to be identified at runtime and loaded dynamically.

9.3.2 Information Requirements of Attribute Providers

To identify a subject an attribute provider needs certain attributes, which refer to the subject non-ambiguously. If the Access Control Product and the attribute provider are closely related, the provider could use the user identifier of the Access Control Product as subject identifier. Unfortunately in most cases this relationship is not the case. Attribute providers will use unique attributes of the user or combinations of several non-unique attributes with small collision probability.

Unfortunately the Access Control Software's subject information does not cover all the information requirements of different providers. To communicate with such providers either the Access Control Product has to query the user for this data or the `DynAdiEntitlementService` use a browser-based protocol to ask the user for these attributes itself. The first version of the `DynAdiEntitlementService` depends on the access control software to send these these data within the `getEntitlements()` request.

Which attributes are needed to authenticate a user, depend on the concrete provider. The following information are likely examples:

¹We give a short introduction to the protocols that are related to the attribute retrieval with fixed providers in Section 6.3.

Social Security Number (SSN): The US SSN is one example of unique identifier assigned by a government organization.

DUNNumber (Data Universal Numbering System): A standard for keeping track of millions of businesses worldwide. It provides unique identifiers of single business entities, while linking corporate family structures together.

Email address: Different providers use the subject's personal e-mail address to identify the subject.

Combination of full name, day of birth and native country: We assume that these data are sufficient to identify a subject without ambiguity.

Subject identifier of the access control software: If the attribute provider is aware of the unique subject identifiers the access control software assigns to authenticated subjects, these can be used, too.

In most cases the access control software has to enrich the subject data handed over with the `getEntitlements()` request with the information the attribute providers need. The DynADI Entitlement Service can utilize browser-based protocols (i.e. a SAML Single Sign-on profile [SAML02b], MS Passport [Mic02a] or IBM Idemix² [CH02]) to retrieve these data on its own.

9.4 Static Class Design

This section contains the class design of the most important classes of the `DynAdiEntitlementService`.

9.4.1 The Class `DynAdiEntitlementService`

The class `DynAdiEntitlementService` provides the interface to the access control software. It organizes the retrieval of the requested `DynAdiContainers` and the generation of the Session corresponding to the request. The class `DynAdiEntitlementService` is responsible for the initialization of the service, the logging and the central data structures.

The instance of the class performs validity tests for all inputs the `DynAdiEntitlementService` gets and checks the data structures for consistency. Further more it analyzes and handles Exceptions of other classes. Any exception thrown by a sub component is caught by this class, enriched with more details about the exception recovery and returned to the access control software.

9.4.2 The `WebServiceAdapter`

On the boundary to the Access Control Software we use the Adapter design pattern (Section 6.4.3) to anticipate future changes and to provide interchangeability of interfaces. We designed the relationship between the Web Service interfaces and the `DynAdiEntitlementService` such that you can bin several Web Service front-ends to our service without having to change the service. We depict the class diagram of this design in Figure 9.5.

²<http://www.zurich.ibm.com/security/idemix/>

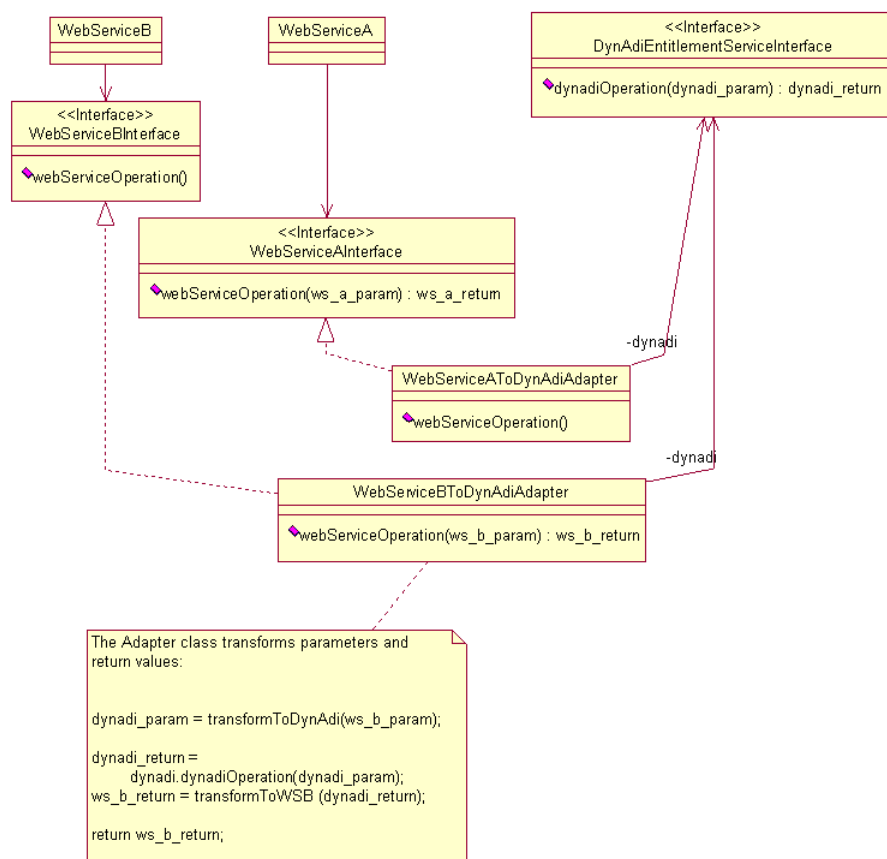


Figure 9.5: Relationship Between the Adapters and the Entitlement Service

Utilizing this solution we accept an overhead of additional method calls and necessary transformations of data-structures: As you can see in Figure 9.5 an Adapter has to compute several transformation within the adaptation of one method call. On the one hand it has to transform the parameters provided by the Web Service to a format compatible with the DynAdiEntitlementService's parameters. On the other hand the Adapter has to transform back the return value of our service to the format the Web Service returns.

In accepting this overhead you gain a lot flexibility. On the one side you can change the Web Service interface easily without changing the interface of the DynAdiEntitlementService itself. On the other side you have the possibility to enhance the service with further (not necessarily Web Service) interfaces for other applications or Access Control Products. Using the Adapter pattern we enhance the reusability of our service.

9.4.3 The Classes Session and SessionTable

For each new `getEntitlements()` request the DynAdiEntitlementService will create a new Session, which holds all information corresponding to that request. All different Sessions of the DynAdiEntitlementService are stored in a data structure called SessionTable. This class manages the access to the different Sessions and finds the Session that corresponds to a certain `session_id`.

The class SessionTable is designed as Singleton, i. e. there is only one instance of this class. The only access point to that instance is static method `SessionTable.getInstance()`. The details of this design pattern are described in [GHJV95] and in Section 6.4.1. We will use it several times in the project.

Note: We will use unique identifier of the Subject of a request as `session_id`. That means that users with the same identifier will be mapped to the same Session. Therefore, the access control software has to authenticate the users and assign an unique identifier.

Each instance of Session generates several DynAdiClients to perform the retrieval of the requested DynAdiContainers. For each required combination of DynAdiProtocol and DynAdiProvider there exists exactly one. The class Session manages the initialization and order of execution of the different instances of DynAdiClient. The Session also distributes the requested `container_type_ids` on the different clients. We depict the relation between Session and DynAdiClient in Figure 9.6.

9.4.4 The Class DynAdiClient

A DynAdiClient is the module, which performs the communication with an attribute provider. It has an instance of class DynAdiProtocol and a reference to a DynAdiProvider. The class DynAdiProtocol contains the information how to run a certain protocol and the client's protocol-specific state. We depict the relation of the classes DynAdiClient, DynAdiProtocol and DynAdiProvider in Figure 9.7:

The instance of DynAdiProtocol is such to say plugged into the DynAdiClient. It serves as Strategy for the question how to communicate with the attribute provider. The DynAdiProtocol gets all information needed to run from the DynAdiClient. See also [GHJV95] or Section 6.4.4 for more details about the Strategy pattern. The major advantage of the use of this pattern is the possibility to replace the protocols easily.

We use a Factory Method (as described in [GHJV95] and Section 6.4.2) to build the different clients. The factory's `generate()` method gets the `protocol_id` and `provider_id` as parameter and generates an instance of class DynAdiClient for that combination.

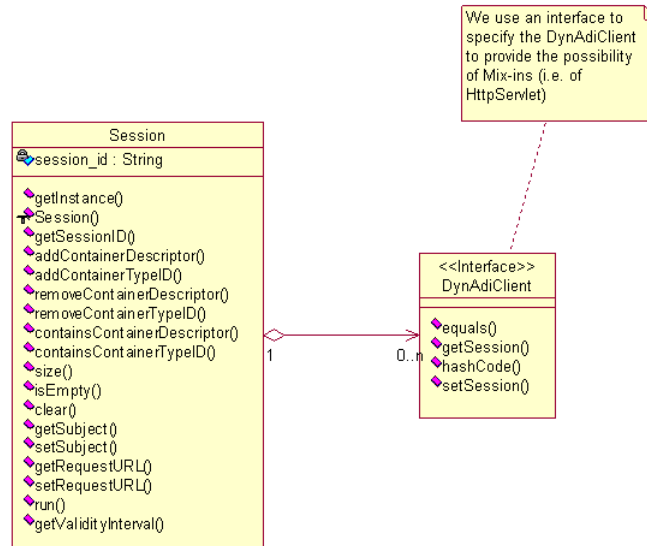


Figure 9.6: Class Design of the Session

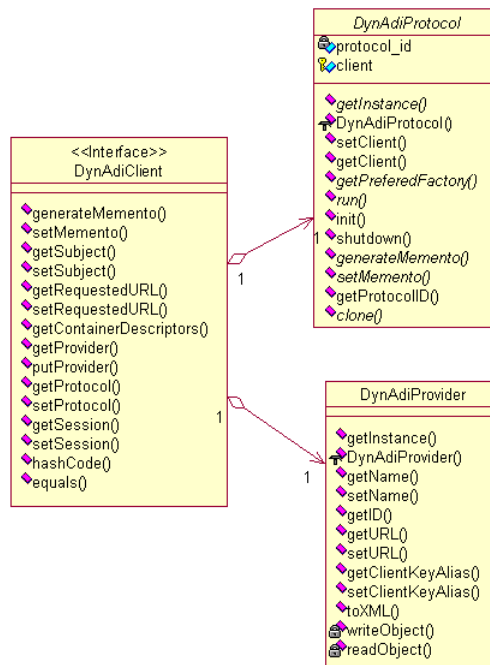


Figure 9.7: Class Design of DynAdiClient and DynAdiProtocol

9.4.5 The Class DynAdiProtocol

A DynAdiProtocol contains the information about a protocol flow. It is stateful and contains all information about the DynAdiProtocol's current state. We use the hierarchy of Figure 9.8 for the protocols:

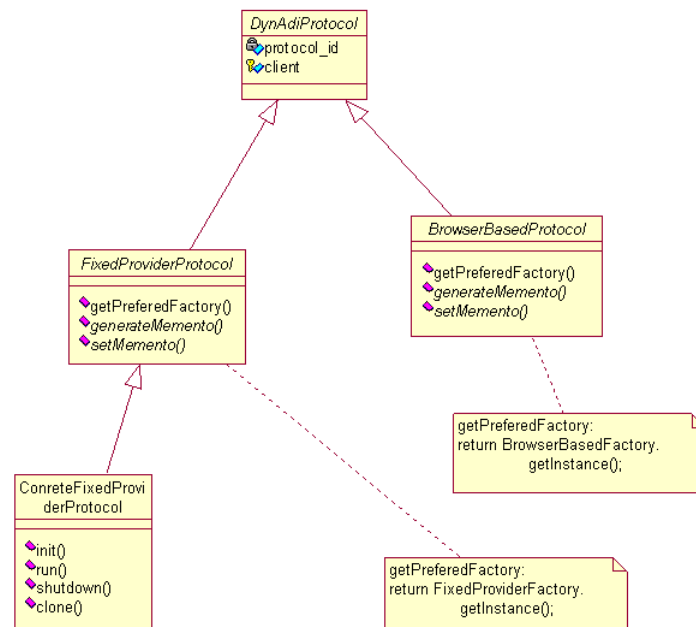


Figure 9.8: Hierarchy of Protocol Classes

9.5 Usage of Design Patterns

9.5.1 Single Instance per ID

Single Instance per ID is one of our own patterns. We designed it for the following purpose. In the design of the DynAdiEntitlementService we face several times the situation that we have one central data table that contains elements referenced by a unique ID. It is crucial for the service that this ID refers non-ambiguously to one single instance of the corresponding element.

As depicted in Figure 9.9 the elements of Product dependent from a certain ProductTable have no public constructor. Instead they contain a static getInstance() method that is responsible for the retrieval of an element of type Product. This method delegates the call to the corresponding instance of ProductTable. As the table knows all instances of the Product it can check the given ID, whether it was explicitly registered before.

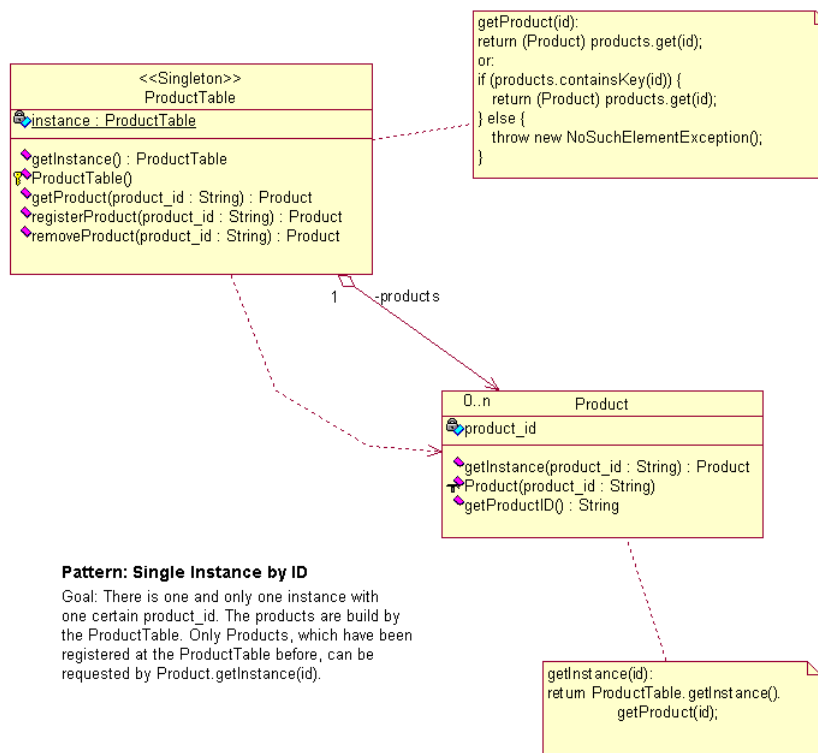


Figure 9.9: Class Diagram of the Single Instance per ID

The ProductTable is member of the same package as the Product and is therefore allowed to call the Product's constructor during the registration. It generates an instance of Product, if the new element is consistent to current data table.

9.5.2 Design of Stateful Protocols

Especially browser-based protocols have usually several states. We use the State design pattern as described in Section 6.4.5 and [GHJV95] to design the state transactions.

We recommend to keep all the Protocol's changeable variables in the Protocol class itself. Then you can use stateless ProtocolState classes that define the Protocol's behavior in a certain state. We depict this design in Figure 9.10 The Protocol delegates method calls to the Protocol methods initialize(), run() and shutdown() to the current state with a reference to itself as parameter. Given this reference the ProtocolState class can access the Protocol's intrinsic state without needing to have a state on its own.

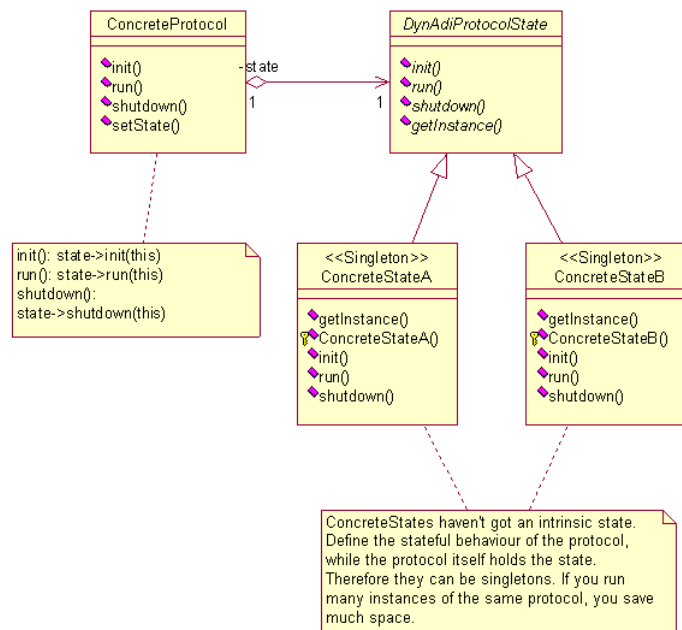


Figure 9.10: Protocol class with stateless ProtocolState

9.6 Dynamic Behavior and Collaboration

We describe some aspects of the dynamic behavior of the entitlement service in this section.

9.6.1 Rough Flow of a getEntitlements() Request

In this section we introduce the request flow of getEntitlements() method call. We present an overview over a request for one container that is retrieved with a fixed-provider protocol. Thus, This is this simplest flow non-trivial flow possible. We depict the calling sequence in a sequence diagram in Figure 9.11 and in a corresponding collaboration diagram in Figure 9.12.

We do not provide all details in these Figures and only consider six objects. The object core is an instance of DynAdiEntitlementService. It receives the request and generates a corresponding Session instance called session. The object factory is responsible for the generation of instances of DynAdiClient. The DynAdiContainerDescriptor instance cdesc contains the data corresponding to the container that is retrieved. The objects client and protocol are instances of DynAdiClient and DynAdiProtocol. They are responsible for the retrieval of the requested container.

The flow has three phases:

1. Creation of the Session. Within this phase the session is instantiated and set up with all data necessary for the attribute retrieval.
2. Distribution of the requested container_type_ids to corresponding clients. Each container_type_id is added to an instance of DynAdiClient that is responsible for its retrieval. If no clients exists yet that is capable of retrieving the container, a new one is created by the factory.
3. The Session runs the different client modules. Within this phase the session initializes, runs and shutdowns the different clients. The clients retrieve the necessary attributes and transform them to containers. After that the session instance decides about a possible redirect.

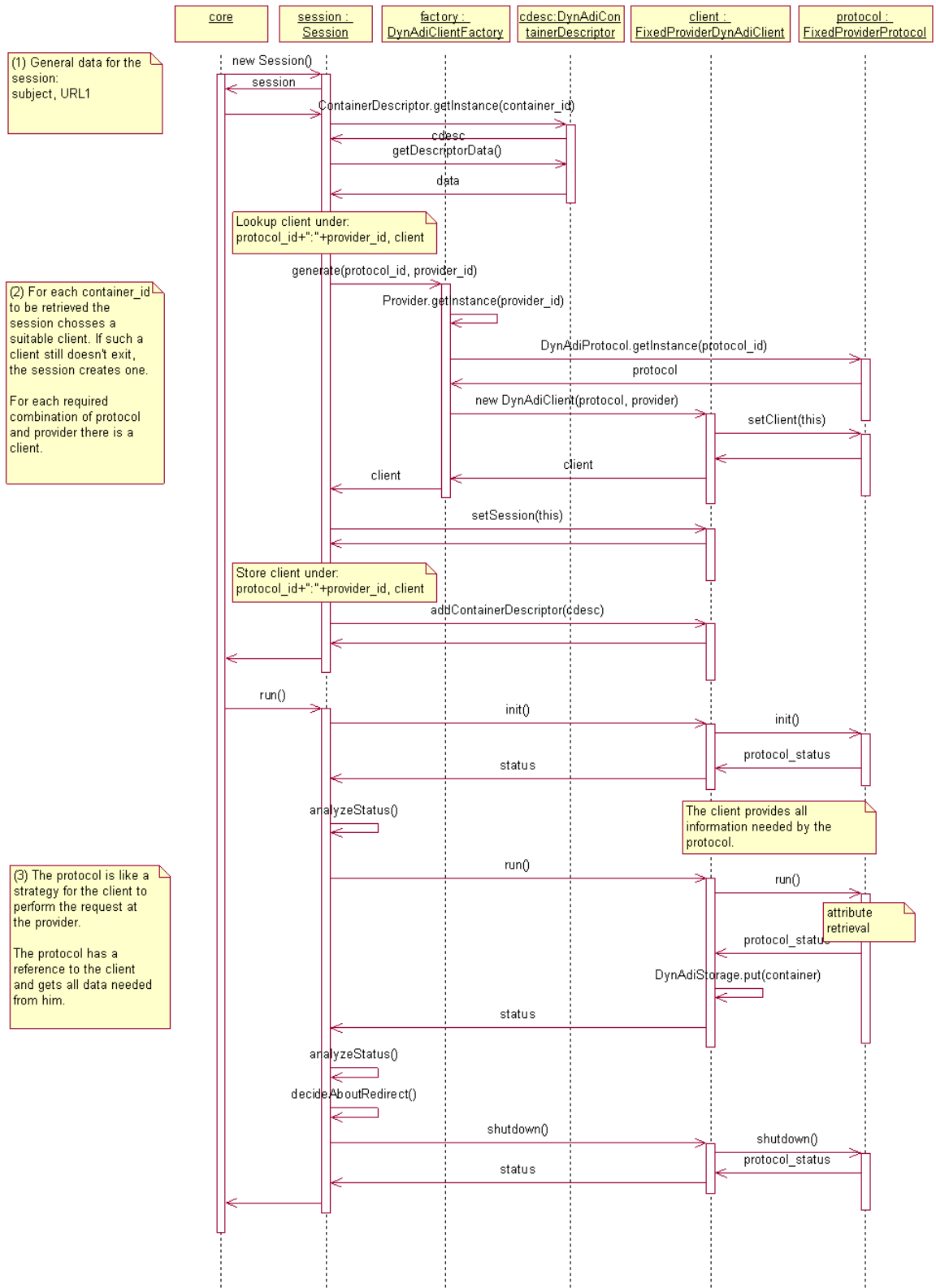
9.6.2 Example Protocol Flow

In this section we depict an example protocol flow for a fixed-provider protocol similar to that one we integrated in the DynAdiEntitlementService. It is a simple request-response protocol. In Figure 9.13 we provide a sequence diagram for this protocol module. Figure 9.14 depicts the corresponding collaboration diagram.

Both figures are concerned about four objects collaborating. The fixed_provider_protocol is the instance of the DynAdiProtocol we use as example. The objects attribute_request and attribute_response are messages send and received during the communication with the provider. The transport object encapsulates the functionality to connect to the DynAdiProvider.

The following steps are typical:

1. Loading of the protocol or provider specific keys. As the communication with the attribute provider's server is usually secured and the messages signed, this is necessary.
2. Generation of a Transport instance that is responsible for the communication infrastructure.
3. Retrieval of the attribute names. The protocol class has to fetch the attribute names from the DynAdiContainerDescriptor and possibly transform them into the attribute names the provider uses.



03/31/2003 Figure 9.11: Sequence Diagram the getEntitlements() Request

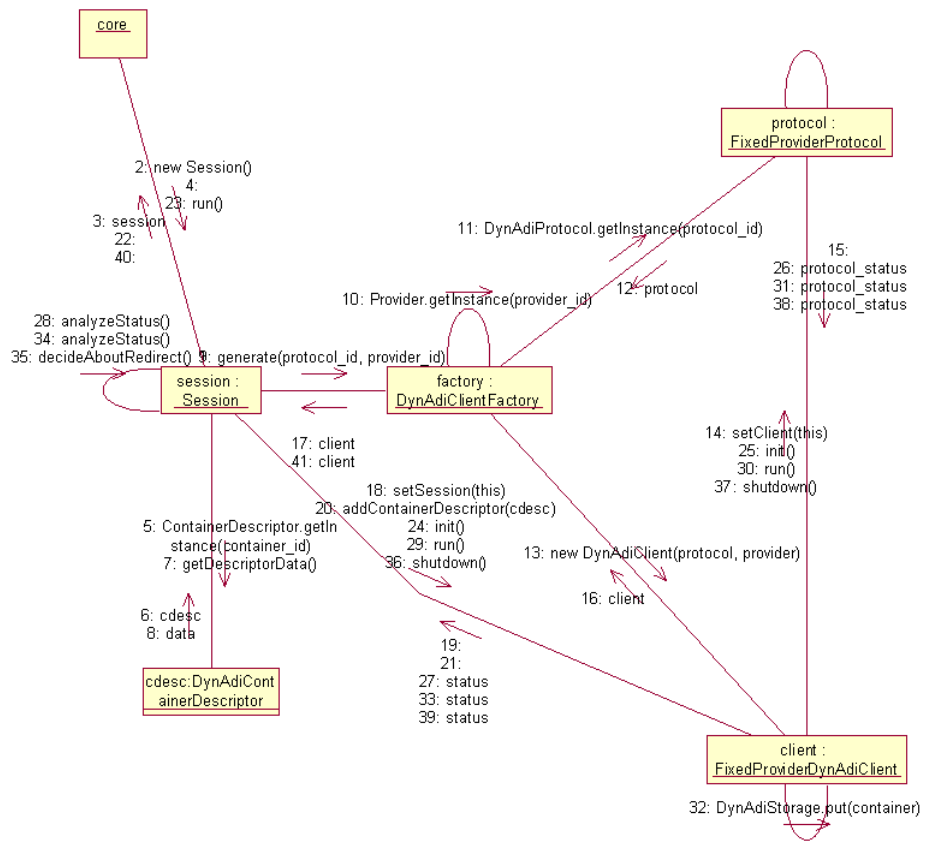


Figure 9.12: Collaboration Diagram the getEntitlements() Request

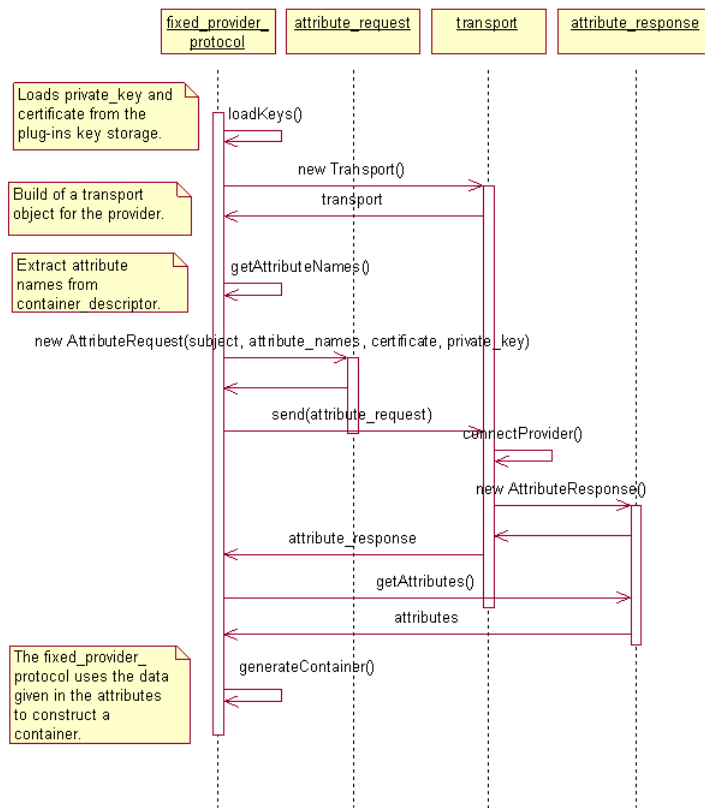


Figure 9.13: Sequence Diagram of a Fixed-provider Protocol

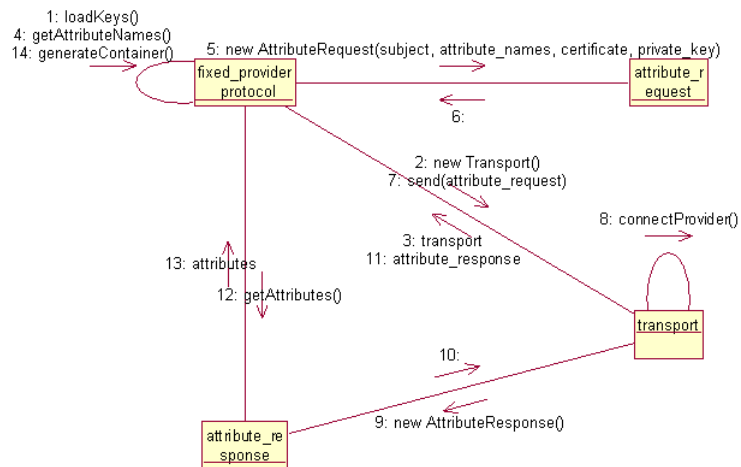


Figure 9.14: Collaboration Diagram of a Fixed-provider Protocol

4. Generation of a `AttributeRequest` object. This object encapsulates the request-specific data such as the required attributes and the subject.
5. Send of the `Request` instance using the `Transport` object. The `send()` usually returns a `AttributeResponse` object corresponding to the request.
6. Retrieval of the attributes from the response.
7. Generation of the `DynAdiContainers`. The protocol instance then creates the instances of `DynAdiContainer` to be returned to the `DynAdiEntitlementService`. The protocol has to transform the attribute names to the Access Control Product's format and to enrich the `DynAdiContainer` instance with non-attribute data such as validity time.
8. At this point the containers can be cached in the `DynAdiStorage` and added to the client.

Chapter 10

Implementation Issues

This chapter contains a few aspects of the implementation and testing phases of our project. Section 10.1 describes the testing methods we applied and technique of exemplary specification. In Section 10.2 we describe the optimization measures we used.

10.1 Testing

10.1.1 Exemplary Specification

One important part of the the testing of our DynAdiEntitlementService are module tests. We test the interface of each important class of the project against its specification. That means we test each method of the class for a correct behavior according its specification. For this task we use the jUnit test suite that we introduce in Section 10.1.2.

To fulfill this we use the technique of exemplary specification as described in [ZS02]. For each important class and each of its methods we worked according to this algorithm:

1. (*Low-level Design*) Write down the interface of the class and the detailed textual specification of its methods and functionality. We do this in writing down the header of the class and all methods together with its specification as JavaDoc comments.

2. Write a black-box test for each method of class, which tests all specified properties, for instance:

Return value: We check whether the method computes correct return values. For this test we change the program state and the inputs systematically and predict the corresponding return value.

Change of state: Tests whether the method changes the program state correctly. For instance, if the method removes all elements of a data structure, the test method checks whether the data structure is empty afterwards.

Side effects: If the method produces additional side effects, they have to be tested, too. For instance if the method writes a file to disk, we check this file for consistency.

Handling of parameters: Check the behavior of the class with different input values. Depending on the concrete situation we use samples of value classes or random values. We classify values according to the properties that make

them valid or invalid on the one side. And according to their influence on the method on the other side. We test with one sample value of each class and sample values of the class limits.

Throw of exceptions: We check whether the method throws the desired exceptions in the specified cases and whether it prevents exception throw in other cases.

Handling of uninitialized or inconsistent state: We change the program state such that the class of the method is in an uninitialized or inconsistent state. The method has to handle this case with a specified exception.

The test case then is the exemplary specification of the corresponding method. It dictates the interface and later behavior of that method.

3. (*Implementation*) Implement the method according to the specification in the test case. Run the test case to check the implementation for correspondence to the specification.
4. (*Testing*) Extend the test case for each method according to implementation details. It is important not to change the test case parts which specify the method. You only refine the test case with the knowledge of the implementation to a white-box test, which takes the concrete program paths in account.

10.1.2 The jUnit Test Suite

We use the jUnit test suite¹ to implement the test cases of our module tests. The suite provides a good infrastructure for the class tests. It contains TestCase classes and Suites that combine single TestCases to whole test beds. It also provides automatic test runners and graphical user interfaces that allow an easy handling of the module testing.

In Figure 10.1 we depict that class structure of the classes TestCase and TestSuite. The ConcreteTestCase contains a method setUp() that initializes the tested class to a well-defined state and provides additional objects needed for the tests. This method is called before of each call of a testMethod(). The tearDown() method is called after each one. It removes all changes possibly made to the tested class and removes all references to generated objects.

The testMethod() itself usually tests one method of the tested class. It makes assertions about the methods behavior and performs tests for these assertions. We describe some of the methods jUnit provides for these assertions in Figure 10.2. If such a method fails, jUnit registers this failure and reports the test case as failed.

10.1.3 Simple Instance of Exemplary Specification

Let's consider a small example of this technique. We specify a method of a data structure that removes one specific element. The method gets one parameter key, which names the key associated with the element. If the data structure does not contain the requested element the method throws an instance of NoSuchElementException. If the key is *null* the method throws an instance of NullPointerException. In other cases the method removes the element from the data structure and returns it. Figure 10.3 contains the JavaDoc specification corresponding to Step 1.

The corresponding jUnit test case is build up as follows. The jUnit methods used are described in Figure 10.2.

¹<http://www.junit.org>

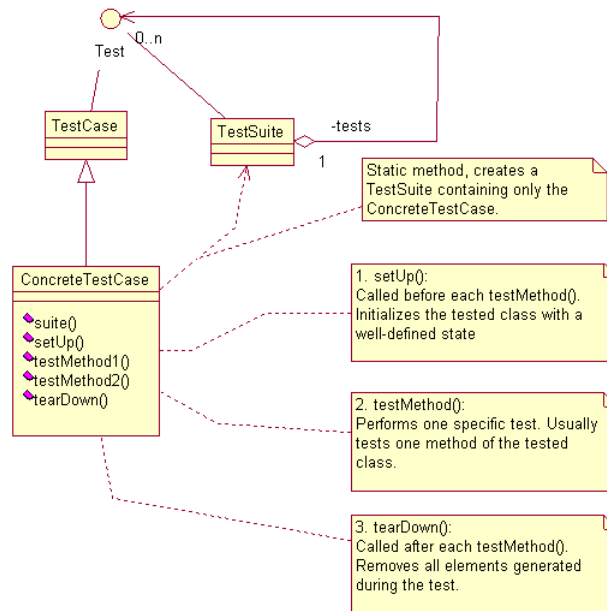


Figure 10.1: Class Diagram of junit TestCase and TestSuite

```

/** Declares the test as failure and outputs the given
    test_failure_message */
public static void fail(String test_failure_message);

/** Checks the given test_condition. If it evaluates to
    false, the method fails with the test_failure_message
    as output. */
public static void assertTrue(String test_failure_message,
    boolean test_condition);

/** Checks whether two objects are the same reference. If
    this is not the case, the method fails with the
    test_failure_message as output. */
public static void assertEquals(String test_failure_message,
    Object expected_object, Object result_object);

```

Figure 10.2: Interfaces of the junit Methods Used

```

/**
 * Removes the element corresponding to the specified key
 * and returns that element.
 *
 * @param key the String key that is assigned to the
 *   element to be removed.
 * @return the element that is removed.
 *
 * @exception NoSuchElementException if the data structure
 *   does not contain an element associated to the key.
 * @exception NullPointerException if the key is null.
 *
 * @post The data structure does not contain the element
 *   corresponding to the given key anymore.
 */
public Object removeElement(String key)
    throws NoSuchElementException, NullPointerException

```

Figure 10.3: JavaDoc Specification of an Element Removal Method

Return value and change of state We remove an element that the data structure contains and test, whether the element was removed successfully, whether only this element was removed and whether the correct element was returned.

```

int expected_size = data_struct.size() - 1;
Object removed = data_struct.removeElement(exist_key_1);
assertSame("The method did not return the correct element",
    exist_element_1, removed);
assertTrue("The requested element was not removed.",
    !data_struct.contains(exist_key_1));
assertTrue("The method removed wrong number of elements.
    Expected new size: " + expected_size +
    " got:" + data_struct.size(),
    expected_size == data_struct.size());

```

Throw of exceptions All given exceptions depend on the parameter values of the method. We hand over a *null* pointer and a key not contained in the data structure to test the correct throw of the exceptions.

```

try {
    data_struct.removeElement(not_exist_key);
    fail("The method did not throw NoSuchElementException
        given a key not in the data structure.");
} catch (NoSuchElementException no_such_element) {
    /* expected exception: OK */
}

try {
    data_struct.removeElement(null);
    fail("The method did not throw NullPointerException
        given null as key");
}

```

```
    } catch (NullPointerException null_pointer) {  
        /* expected exception: OK */  
    }  
}
```

10.1.4 Advantages of Exemplary Specification

As you can see in the example above, the designer needs to write even for small methods large test cases. For most methods of our `DynAdiEntitlementService` the test case corresponding to a method is much larger than the implementation of the method itself. Therefore the exemplary specification in the low-level design phase consumes much time, perhaps more time than the implementation itself.

However we prefer this method. It has the following advantages over non-formal specification.

It forces to think twice: The designer has to know exactly what the designed method or class is supposed to do to write good test cases. The procedure to write the test case before the implementation forces the designer to analyze the functionality of the class systematically.

Explicit specification: The technique forces the designer to write down all specifications explicitly. According to our experience the designer will notice, when he cannot answer a certain question about a method's or class' behavior.

Efficient implementation: Because of the detailed low-level design, the designer has a very clear picture of the functionality of the classes. If the implementer is involved in the low-level design, he will be able to implement the designed classes very fast.

Instant feedback: The technique provides test cases for all methods before their implementation. Therefore in running the test case the implementer can get instant feedback whether a method fulfills the specification or not.

10.2 Optimization

In this section we describe the methods that we applied to improve the quality of service of the `DynAdiEntitlementService`. We provide different optimization measures that decrease the memory usage and number of reinstantiations of the service.

10.2.1 Optimization According to Strict Rules

Two of the most important articles concerned about Java optimization, one by Jonathan Hardwick [Har97] and the other one by Doug Bell [Bel97] refer to a famous quote of Donald Knuth: "Premature optimization is the root of all evil." Given this truth we had to decide whether to optimize at all and which rules to follow within this process.

At the end of the implementation phase we performed some stress tests with the `DynAdiEntitlementService`. Within these tests we sent a large amount of `getEntitlements()` requests to the service and tested its behavior. The test results show two different problems. On the one hand the service produced large response times for some requests presumably because of the activated Garbage Collection. On the other

hand the service ran frequently out of memory because of the large amount of objects it produced. Both problems would influence the quality of service of the entitlement provision negatively or make the `DynAdiEntitlementService` inapplicable for high-load environments.

Having this result in mind we decided to optimize the `DynAdiEntitlementService` towards a better memory usage. We did this according to well-defined rules:

No sacrifice of architecture for performance: We did the design of the `DynAdiEntitlementService` without making compromises for optimization considerations. We followed the principles of modularization and information hiding as well as the anticipation of future change. We tried to get a clear design with good encapsulation. Only after the design and the core of the implementation phase we started the optimization work. This approach follows M. A. Jackson's two optimization rules [Blo01]: "Don't do it." and "(for experts only). Don't do it yet."²

We followed the 80/20 rule: This rule says that 80 percent of a program's execution time is spent in 20 percent of its source code. This implies a rigorous analysis of the program's run to get aware of the important 20 percent. We used IBM's `jInsight` profiler³ to get a clear picture, what the `DynAdiEntitlementService` does and when.

"Before" and "after" Profiles: We used the `jInsight` profiler for a analysis before our optimization work and also analyzed the program profile afterwards. This allowed a direct evaluation of our optimization measures.

Clear objective: We do not want an overall optimization of the `DynAdiEntitlementService` involving source code changes over the whole product. We want to provide a few effective and clear designed measures to solve the quality of service problem above.

10.2.2 Performance Bottlenecks of the `DynAdiEntitlementService` Prototype

Given these rules we analyzed the service's program run with the `jInsight` profiler to find the most important spots of object generation within the project. We concentrated on the program slice executed with `getEntitlements()` method call as this method is executed mostly on the long run. We there figured out the following issues:

String concatenation: Through the whole run of the `getEntitlements()` call, the `DynAdiEntitlementService` concatenates a lot of Strings using the standard concatenation operator '+'. Unfortunately the concatenation of two Strings `str_a` and `str_b` results in the following operation:

```
str_a + str_b  $\iff$  {  
    StringBuffer = new StringBuffer();  
    buffer.append(str_a);  
    buffer.append(str_b);  
    return buffer.toString();  
}
```

²The first rule advises not to do optimization at all. The second rule states that only experts are able to do so. For them the rule recommends not to do optimization in the early project phases.

³<http://www.alphaworks.ibm.com/tech/jinsight>

This operation is really inefficient and produces unnecessary new objects (especially a `StringBuffer` thrown away afterwards).

Redundant construction of reusable objects: Within a `getEntitlements()` call the `DynAdiEntitlementService` generates new instances of several classes of its infrastructure. It generates an instance of `Session` and for each protocol and provider combination one instance of `DynAdiClient`, `DynAdiProtocol` and `DynAdiProvider`.

Both cases have in common that the objects are in principle reusable and that their recreation is problematic. On the one hand the `Session` instance is quite large. Its generation implies the construction of other Java data-structures. On the other hand as the Access Control Product usually requests several containers, it might be the case that the entitlement service generates many instances of client, protocol and provider.

To solve these two problems we used the following measures:

String externalization: For the main program flow, a successful `getEntitlements()` request we externalized all Strings. Instead of generating a new `String` each time we needed one, we generated all pre-computable Strings at initialization time and assigned them to final variables.

Refined concatenation: For the classes that use `String` concatenation at most, we provided a new mechanism for `String` concatenation. The concatenation is delegated to a method that uses a shared `StringBuffer`.

Utilization of resource pools: We use the pool design pattern described in Section 10.2.3 to generate and manage pools of `Session` and `DynAdiClient` instances. These pools are generated during the initialization phase and filled with instances of the corresponding resource. The pools grow exponentially during run time if the number of needed elements exceeds the pool size.

We did not use the resource pools for `DynAdiProtocol` and `DynAdiProvider`: As the protocols are implemented by the customers we could not guarantee that they fulfill the requirements of the resource pool. The providers are as such stateless and need not to be recreated at all.

10.2.3 Resource Pool Design Pattern

In this section we describe the Resource Pool pattern we used to solve the optimization problems. It was inspired by different proposals in the Java community such as [BW98] and [Dav98]. Our variant of this pattern utilizes a class called `PoolManager` to store and manage the pool elements.

The `PoolManager` keeps track of the objects in use and doubles the size of the pool, if not enough elements are available at one time. We combine the `PoolManager` with the prototype pattern to fill the element pool: At construction time the class `PoolManager` receives a prototype of the element to store. Whenever the pool grows, it clones this object to generate further instances of this class. This has the advantage that you can put objects with precisely defined state in the pool.

The `PoolManager` cooperates with an interface called `PoolElement`. This element type guarantees the the object stored in the pool is cloneable and provides a method called `clearInternalState()`. The latter method has the responsibility to set the element back to its original well-defined state.

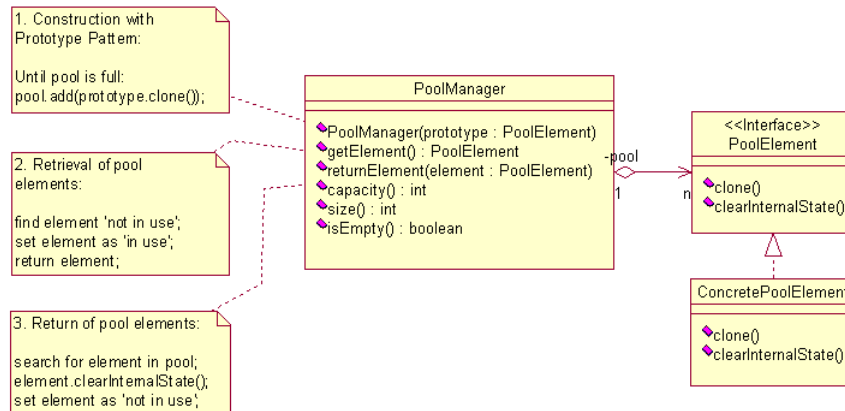


Figure 10.4: Class Diagram of the Design Pattern Resource Pool

When the `getElement()` method is called the `PoolManager` extends the pool, if it is empty. It searches for an unused element, sets it as used and returns it. If the `returnElement()` method is called, the `PoolManager` calls the element's `clearInternalState()` method and sets it as unused.

In Figure 10.5 we depict the application of this pattern to the class `SessionTable`. Within this figure you can see the big advantage of this pattern: It fits naturally into our Single Instance per ID pattern of Section 9.5.1 and requires only minimal changes to the original design. We only replaced the new `Session()` calls of the `registerSession()` methods with the `getElement()` calls to the pool manager and added the `returnElement()` call to the `removeSession()` methods.

10.2.4 Usage of the jInsight Profiler

In this section we introduce the results of our profiling analysis. We depict a profile for the object instantiation in Figure 10.6. The figure contains a profiling analysis of a `getEntitlements()` request. In the vertical dimension you see the time line. In the horizontal dimension the number of active methods on the method stack. Each small horizontal line is equivalent to a method or constructor call.

We have most details of the profile to stress two statements:

Phases of the `getEntitlements()` call: Within this profile we have highlighted four phases: The first one is the construction of `Session` instance. In the second phase the `container_type.ids` are added to the `Session` object and distributed to different clients. The third phase contains the `run()` call of the `Session` object. The last phase contains the call of the `outputString()` method of the class `XMLOutputter`.

These last three phases are equivalent to the most time-consuming methods named in Figure 10.9. The profile of Figure 10.6 approves the statement of that tabular.

Points of Instantiation: We have highlighted some points within the profile where the original version of our service instantiates new objects and the optimized variant does not.

This approves the statement of our object instantiation estimates of Figure 10.7.

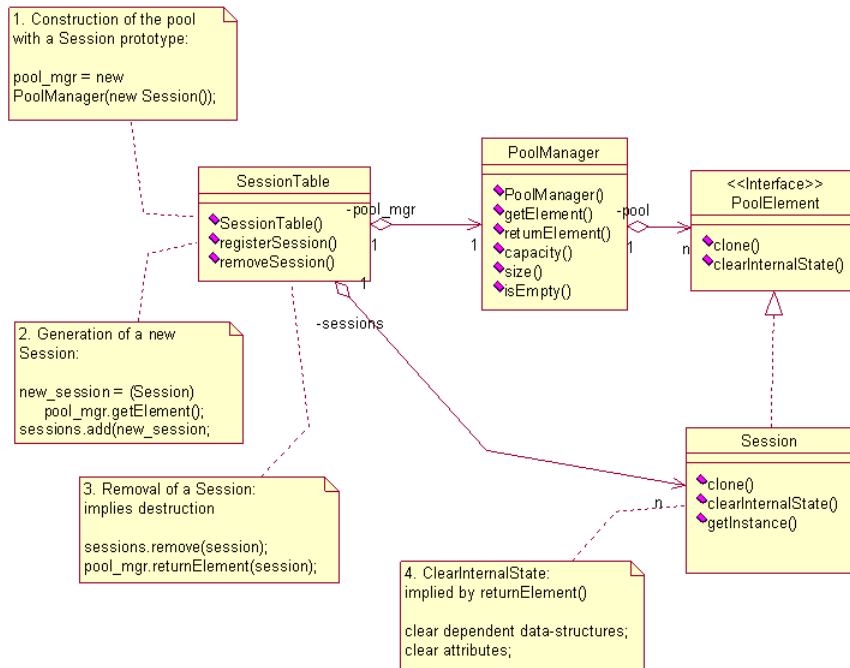


Figure 10.5: Class Diagram of the Resource Pool Pattern to the SessionTable

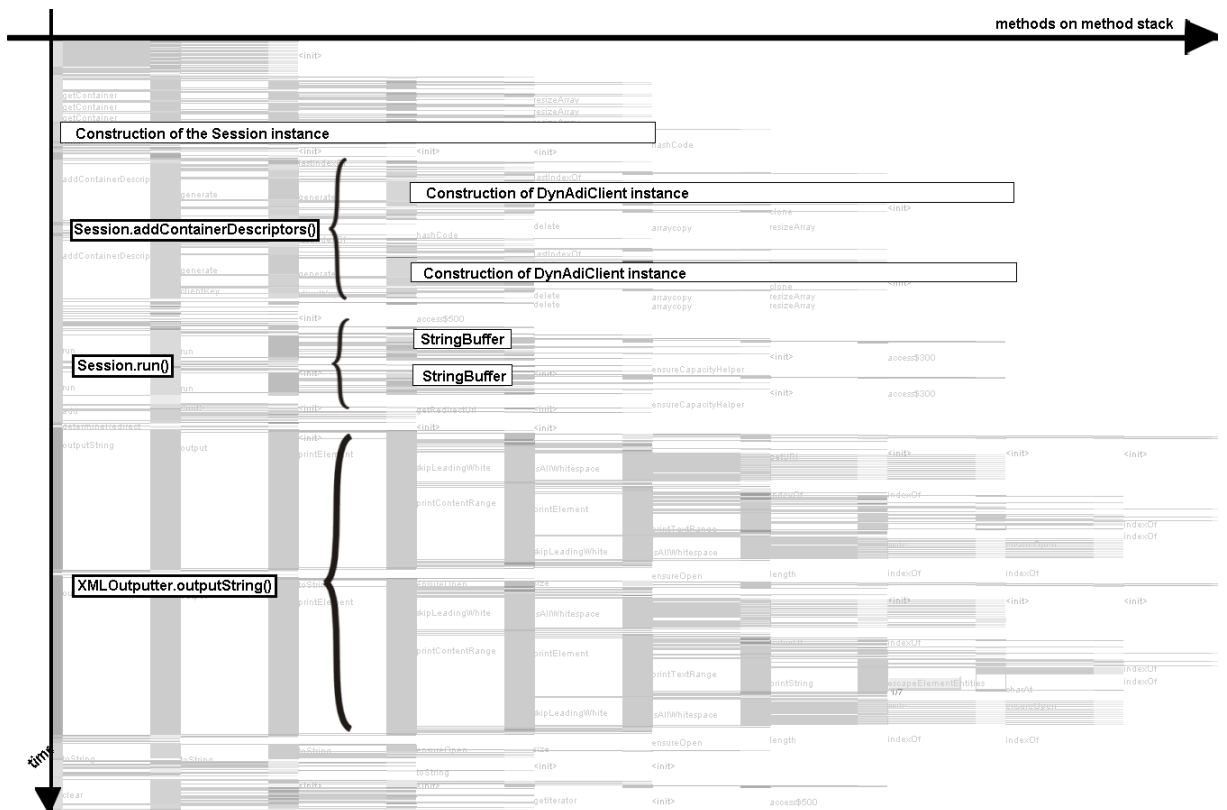


Figure 10.6: Profiling Analysis of the Object Instantiation

10.2.5 Performance Improvements

Within this section we present the results of the performance tests and profiling before and after the optimization. We begin with the memory usage of the DynAdiEntitlementService.

The original version of the service generates one instance of Session and number of combinations of provider and protocol instances of DynAdiClient. For n getEntitlements() requests we get a linear number of instances of Session and super-linear many instances of DynAdiClient, StringBuffer and String. We depict the detailed complexity estimations in Figure 10.7. In this estimate l_{prov} and l_{prot} are the number of providers and protocols specified in the service's configuration files, whereas l is the number of clients used for a single request and n the number of getEntitlements() calls.

The new version of the service has resource pools for the classes Session and DynAdiClient. Though the service may enlarge the pool, the amortized number of instances of both classes is still constant. Apart from creations of the protocol modules, the service does not create any new instances of String and StringBuffer in the critical methods. So the number of these instances is now at most linear.

Class	Original		Optimized	
	Instances	Estimate	Instances	Estimate
String	$\sim 15 + 8 \cdot l$	$O(l_{prov} \cdot l_{prot} \cdot n)$	0	$O(1)$
StringBuffer	$1 + l$	$O(l_{prov} \cdot l_{prot} \cdot n)$	1	$O(n)$
Session	1	$O(n)$	0	$O(1)$
DynAdiClient	l	$O(l_{prov} \cdot l_{prot} \cdot n)$	0	$O(1)$

Figure 10.7: Complexity Estimates for the Object Instantiation

As the management of the object pools can be expensive, the optimization measures may have negative influence on the running time of the service. We have to check that we do not buy the reduction of memory consumption with a worse response time.

Figure 10.8 depicts response time measurements in milliseconds for a scenario with a protocol that reads container from disk. Figure 10.9 contains the utilization of the most used sub-methods of the getEntitlements() method. The performance of the optimized version is slightly better than that of the original one. As our objective was the reduction of the memory usage, this is sufficient.

Requests	Original			Optimized		
	Cumulative	Average	Max	Cumulative	Average	Max
1,000	1392 ms	1.392 ms	220 ms	1392 ms	1.392 ms	210 ms
10,000	4006 ms	0.4 ms	260 ms	3826 ms	0.383 ms	261 ms
100,000	21070 ms	0.21 ms	261 ms	21010 ms	0.21 ms	260 ms
1,000,000	182302 ms	0.182 ms	250 ms	179998 ms	0.18 ms	260 ms

Figure 10.8: Performance Measurements for the Service (4 containers/request)

10.3 Metrics for the DynAdiEntitlementService

In this section we provide some metrics for the DynAdiEntitlementService. We used two different tools to determine these metrics. On the one hand JavaNCSS⁴, a source

⁴<http://www.kclee.com/clemens/java/javancss/>

Method	<i>Original</i>		<i>Optimized</i>	
	Cumulative	Percentage	Cumulative	Percentage
XMLOutputter.outputString()	150502 ms	34.7 %	140194 ms	37.7 %
Session.run()	99159 ms	22.9 %	57314 ms	15.4 %
Session.addContainerTypeID()	46744 ms	10.8 %	46773 ms	12.6 %
checkDynAdiStorage()	25164 ms	5.8 %	22652 ms	6.1 %
shutdownSession()	19084 ms	4.4 %	19147 ms	5.2 %

Figure 10.9: Sub-methods of the getEntitlements() Method

measurement suite for Java, which provides a so-called NCSS metric, the number of classes and methods as well as the number of JavaDoc statements. On the other hand we used LOCC⁵, a size measurement tool, which provides physical size like lines of code.

We consider the following data:

Number of packages, classes and methods: Reflects the granularity of our class design. (Tool: JavaNCSS)

Lines of Code (LOC): The number of physical source code lines. (Tool: LOCC)

Non Commenting Source Statements (NCSS): Number of Java statements and declarations. Approximately the number of semicolons and opening curled brackets. (Tool: JavaNCSS)

Number of Formal JavaDoc Comments (JavaDocs): Number of formal JavaDoc comment blocks. This does not mean the number of lines or comment tags, but only the number of formal documented elements. (Tool: JavaNCSS)

Lines of JavaDoc Comments (LOJ): The number of physical JavaDoc comment lines. We used an own tool to determine this value. It matches the begin and end of JavaDoc comment blocks and counts the comment lines.

While Figure 10.10 contains the metrics for the DynAdiEntitlementService, Figure 10.11 depicts the metrics for the corresponding junit Test Project. Figure 10.12 describes the metrics for the whole project.

The Figures are designed as follows: Each row contains a set of metric values per granularity. While the first row contains the absolute metrics for the whole project, the next three lines describe the average metrics per Package, Class and Method.

Per	Packages	Classes	Methods	LOC	NCSS	JavaDocs	LOJ
Project	20.00	99.00	776.00	8498.00	5754.00	642.00	5327.00
Package		4.95	38.80	424.90	287.70	32.10	266.35
Class			7.84	85.84	58.12	6.48	53.81
Method				10.95	7.41	0.83	6.86

Figure 10.10: Metrics for the DynAdiEntitlementService alone

We deduce the following statements from these metrics:

- With an average length of 7.41 statements we have relatively small, clear arranged methods.

⁵<http://csdl.ics.hawaii.edu/Tools/LOCC/LOCC.html>

Per	Packages	Classes	Methods	LOC	NCSS	JavaDocs	LOJ
Project	10.00	35.00	295.00	3784.00	2739.00	211.00	2686.00
Package		3.50	29.50	378.40	273.90	21.10	268.60
Class			8.43	118.11	78.26	6.03	76.74
Method				12.83	9.28	0.72	9.10

Figure 10.11: Metrics for the jUnit Test Suite of the DynAdiEntitlementService

Per	Packages	Classes	Methods	LOC	NCSS	JavaDocs	LOJ
Project	20.00	134.00	1071.00	12282.00	8493.00	853.00	8013.00
Package		6.70	53.55	614.10	424.65	42.65	400.65
Class			7.99	91.66	63.38	6.37	59.80
Method				11.47	6.37	0.80	7.48

Figure 10.12: Metrics for the Whole DynAdiEntitlementService Project

- The number of source code lines to the number of JavaDoc comment lines relates more than three to two.
- About 80 % of the methods of the DynAdiEntitlementService are documented with JavaDoc comments.
- For about one third of the classes of the DynAdiEntitlementService exist TestCase classes in the jUnit test suite.

Chapter 11

Conclusion and Future Work

In this thesis we followed two goals: On the one hand the protocol analysis of the OASIS Security Assertion Markup Language (SAML). On the other hand the design and implementation of an entitlement service.

In Part II we designed and implemented the so-called DynAdiEntitlementService. This Web Service does attribute retrieval dynamically and on demand by utilization of fixed-provider and browser-based protocols. It allows to exchange the Web Service interface to an Access Control Product as well as the protocol modules that connect to attribute providers. According to the integration and acceptance tests the DynAdiEntitlementService is robust and reliable.

With the DynAdiEntitlementService we provide an attribute function for the middle-ware. As its WebService interface is exchangeable and the abstract interface of the service generic, it is usable for various applications. By using the DynAdiEntitlementService it is possible to externalize attribute provision to a configurable module with only marginal change of the Access Control Product.

We conclude that the utilization of extern modules for attribute retrieval is a good approach for Access Control Products. It helps to maintain clear interfaces and separation of interests.

For the DynAdiEntitlementService, we want to integrate important browser-based protocols like SAML SSO or BBAE. It is interesting for us to provide a set of protocol modules that provide various data used in context-based access control. Additionally, we want to integrate a more flexible generic Session management that can utilize the sessions an Access Control Product generates.

Appendix A

Detailed Requirements

We organize this appendix chapter according to the structure of a requirements document¹ as given in [ZS02] and [Bal96]. While Chapter 7 describes the general problem we address, in this chapter we show the explicit goal of the project, the details of the product use and environment, its functionality and persistent data. In Chapter 9 we introduce the high-level design corresponding to the project.

A.1 Goals

In this section we describe the abstract goals of the project. According to the goal structure of [ZS02] and [Bal96], we distinguish must-have criteria, want-have criteria and non-goals. The must-have criteria have to be fulfilled in order to finish the project successfully. Want-have criteria are nice to have, but are not mandatory. In the section non-goals we show which objectives are excluded from the project.

A.1.1 Must-have Criteria

- Reliable and robust dynamic retrieval of Access Decision Information requested by an access control software.
- Communication with DynAdiProviders. Use of fixed-provider DynAdiProtocols. In this setting the DynAdiEntitlementService retrieves the Access Decision Information without user interaction from a fixed provider.
- Integration into the existing access control software as plug-in, preferably without or with small changes of the access control product. The service has to be delivered as XML based WebService.

A.1.2 Want-have Criteria

- Communication with browsers. Use of browser-based DynAdiProtocols.
- Interfacing to federated protocols for obtaining ADIs.
- Modularity and interchangeability of DynAdiProtocols.

¹In German called "Pflichtenheft"

- Protocol independence. As the development and standardization of federated DynAdiProtocols has not been finished, adapting to a new such protocol should only require internal changes in the DynAdiEntitlementService.
- Reusability of the DynAdiEntitlementService in other products.
- Local short-term caching of DynADI for better efficiency and limiting costs if the DynADI provision is billed on a per-use basis.
- Extensibility to long-term storage of received DynADI, in particular in directions where the Web-Access Control Product retrieves ADI.

A.1.3 Non-Goals

- Graphical user interface.
- Complex DynAdiStorage that goes beyond simple caching in memory

A.2 Product Use

- Entitlement service for an existing access control software. It is supposed to provide a WebService with an interface similar to the access control product's entitlement retrieval interface. It shall be possible to replace the interface of the WebService easily. Therefore the service is usable by other access control or enforcement products, too.
- Addressed users: Professional users of a given access control product.

A.3 Product Environment

Here we show which environment is needed to run the DynADI Entitlement Service.

A.3.1 Hardware

- No relevant requirements to run the DynAdiEntitlementService.
- In high-traffic environments the application server hosting the DynAdiEntitlementService requires a large amount of RAM.

A.3.2 Software

- An access control product enhanced with a WebService client for the DynAdiEntitlementService and the capability to return redirects.
- An application server capable of hosting WebServices or IBM EAR Servlet Archives. (For instance the IBM WebSphere Application Server, Apache Tomcat or Mind Electric GLUE STD)
- For a more customized deployment of the WebService we recommend to use the IBM WebServices Toolkit (IBM WSTK).

- SUN Java JDK 1.4 or IBM JDK 1.3.1
- Different libraries
- Protocol specific libraries and DynAdiProtocol modules.

A.3.3 Orgware

- Modification of the access control software.
- Deployment of the new version of the access control software.
- Deployment of the DynAdiEntitlementService to an application server.
- Configuration of the DynAdiEntitlementService and of the corresponding XML data tables.

A.4 Product Function

In this section we consider the desired functions of the product. We use **bold function numbers** (such as **[RF110]**) to mark functions of a prototype supporting only fixed-provider queries. These functions suffice to fulfill the must-have criteria of Section A.1. In Figure A.2 we show all functions organized as tree. In Figure A.1 we depict the tree of must-have functions. Note that the general structure of both trees is similar.

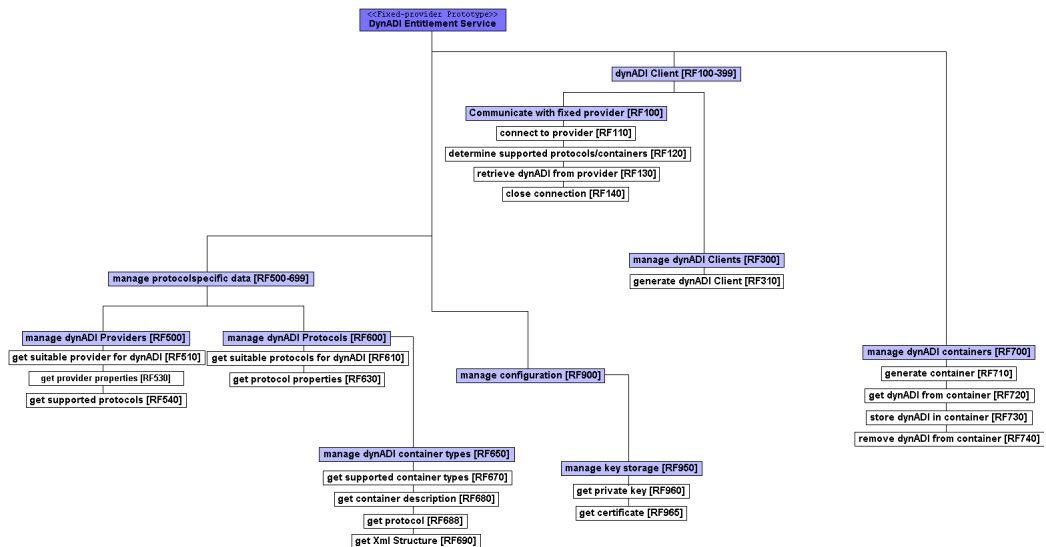


Figure A.1: Tree of Must-have Functions of the DynAdiEntitlementService

DynADI Client [RF100-399] We consider the communication functions of the DynAdiEntitlementService first, which is performed by different DynAdiClients. The design of the corresponding class is described in Section 9.4.4.

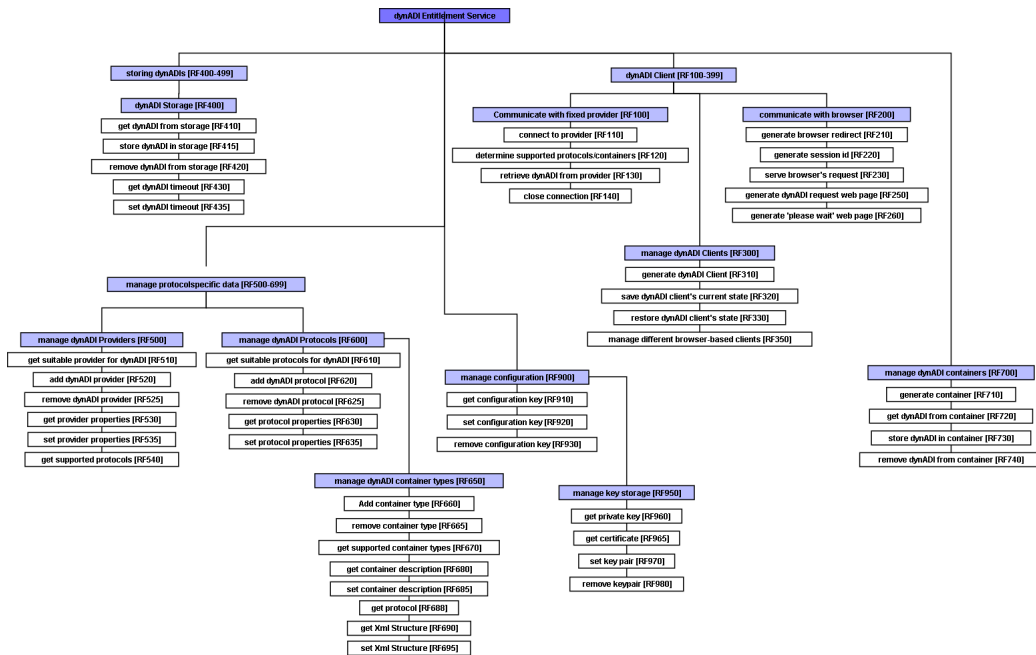


Figure A.2: Function Tree of the DynAdiEntitlementService

- **Communicate with fixed provider [RF100].**
The communication with a fixed provider is the most important function of the DynAdiEntitlementService. The service directly connects the attribute provider's server and tries to retrieve the desired DynADIs.
 - **Connect to provider [RF110]**
The DynAdiEntitlementService connects the provider's server and opens a secure communication channel.
 - **Determine supported protocols and containers [RF120]**
The plug-in determines the list of protocols/DynAdiContainers supported by a DynAdiProvider.
 - **Retrieve DynAdi from provider [RF130]**
The DynAdiEntitlementService chooses a suitable protocol from its protocol table. It runs the protocol with the attribute provider's server in order to retrieve the desired DynADIs.
 - **Close connection [RF140]**
- **Communicate with browser [RF200]**
The DynAdiEntitlementService runs a browser-based protocol to retrieve DynADIs. It redirects the browser focus of the user to his DynAdiClient or a third-party source site.
 - **Generate browser redirect [RF210]**
The DynAdiEntitlementService generates a redirect to lead the browser focus to his DynAdiClient or another source site.
 - **Generate session id [RF220]**
The service generates a unique session id for a DynAdiClient.
 - **Serve browser's request [RF230]**
The DynAdiClient receives the browsers request. It identifies the client through the session id.

- Generate DynADI request web page [RF250]
The DynAdiClient generates a web page to retrieve DynADIs or further information from the user.
- Generate 'Please wait' web page [RF260]
The DynAdiClient generates a web page to show the user the progress of the determination of the desired DynADI.
- Manage DynAdiClients [RF300]
For the use of browser-based protocols it is necessary to manage the different DynAdiClients generated by the DynAdiEntitlementService.
 - Generate DynAdiClient [**RF310**]
For each protocol/provider combination used in a request a DynAdiClient is created. This function is required for the fixed-provider setting, too.
 - Save DynAdiClient's current state [RF320]
For browser-based protocols which need to save the DynADI in their DynAdiClient's state it is necessary to save the state of the client. By using this function you do not need to save the full instance of DynAdiClient in memory, but a small token of its state information.
 - Restore DynAdiClient's state [RF330]
Restore the DynAdiClient with a state token previously saved.
 - Manage different browser-based clients [RF350]
If there are different browser-based protocols in one entitlement request, the different browser-based DynAdiClients compete for the browser focus. The DynAdiEntitlementService has to manage the sequence of their access to the browser focus.

Storing DynADI [RF400-499] It is desirable to store the DynADI for caching purposes. For browser-based protocols it is even mandatory to save the DynADI.

- DynADI Storage [RF400]
The short-term storage of DynADI is mandatory for different protocols. The DynADI are stored temporary in the DynAdiStorage.
 - Get DynADI from storage [RF410]
The DynAdiEntitlementService retrieves a requested ADI from the DynAdiStorage.
 - Store DynADI in storage [RF415]
 - Remove DynADI from storage [RF420]
 - Get DynADI timeout from storage [RF430]
 - Set DynADI timeout [RF435]

Manage protocol-specific data [RF500-699] The DynAdiEntitlementService uses different protocols, providers and container types. The plug-in has to manage them and the relations between them.

- Manage DynAdiProviders [**RF500**]
The DynADI plug-in is supposed to support several DynAdiProviders. It manages a table of the different providers.

- Get suitable provider for DynADI [**RF510**]
The DynAdiEntitlementService looks for a DynAdiProvider which may provide a requested DynAdiContainer.
- Add DynAdiProvider [RF520]
The plug-in adds a DynAdiProvider to its provider table.
- Remove DynAdiProvider [RF525]
- Get provider properties [**RF530**]
- Set provider properties [RF535]
- Get supported protocols [**RF540**]
Get the protocols that are supported by a provider.
- Manage DynAdiProtocols [**RF600**]
The DynAdiEntitlementService has to use several protocols to retrieve DynADI.
 - Get suitable protocols for DynADI [**RF610**]
 - Add DynAdiProtocol [RF620]
 - Remove DynAdiProtocol [RF625]
 - Get protocol properties [**RF630**]
 - Set protocol properties [RF635]
- Manage DynAdiContainer types [**RF650**]
The access control software uses the container types as parameter for the getEntitlements request. The DynADI Entitlement Service manages the different container types to fulfill the requests.
 - Add container type [RF660]
 - Remove container type [RF665]
 - Get supported containers [**RF670**]
 - Get container description [**RF680**]
A container description contains all data used to manage a container type (i.e. protocol used to retrieve the container, XML structure with the attributes it contains, ...)
 - Set container description [RF685]
 - Get protocol [**RF688**] A container type is bound to a special DynAdiProtocol which is used to retrieve the container's data.
 - Get XML structure [**RF690**] The container description shows how the container's payload will be structured.
 - Set XML structure [RF695]
- Manage DynAdiContainers [**RF700**] The DynAdiEntitlementService has to manage the different DynAdiContainers it produces for the access control software.
 - Generate container [**RF710**]
 - Get DynADI from container [**RF720**]
 - Store DynADI in container [**RF730**]
 - Remove DynADI from container [**RF740**]

Configuration Management There are several protocol independent information that have to be managed. On the one hand there are general configuration keys of the DynADI Entitlement Service, on the other hand there is a central key storage.

- Manage configuration [RF900]
 - Get configuration key [RF910]
 - Set configuration key [RF920]
 - Remove configuration key [RF930]
- Manage key storage [RF950]

The DynAdiEntitlementService communicates with several parties over secure channels. It has to store private keys and certificates used to identify itself.

 - Get private key [RF960]
 - Get certificate [RF965]
 - Set key pair [RF970]
 - Remove key pair [RF980]

A.4.1 Rough State Chart

In Figure A.3 we show a state chart for the DynADI Entitlement Service itself. We concentrate on its functions and consider the DynAdiClient and the DynAdiStorage as black boxes.

A.4.2 Data Flow

In this section we show the data flow for the two cases fixed-provider and browser-based protocol. In Figure A.5 we consider the data flow for a fixed provider, in Figure A.6 that of a browser-based DynADI retrieval. We use the notation of deMarco for data flows as described in [GJM91] and [ZS02] (see also the legend in Figure A.4).

A.5 Product Data

This section contains the main data of the product that is permanently saved. It describes the different configuration files used.

DynADI Plug-in Configuration File (Java Properties Format) [RD100]

This file stores the general configuration information for the DynAdiEntitlementService. It contains general information for the service itself, the DynAdiStorage and DynAdiClients.

Protocol Table (XML format) [RD200]

In this file the information about the installed protocols and the containers they may retrieve are stored.

Provider Table (XML format) [RD220]

This XML file contains the data needed to communicate with the attribute providers' servers.

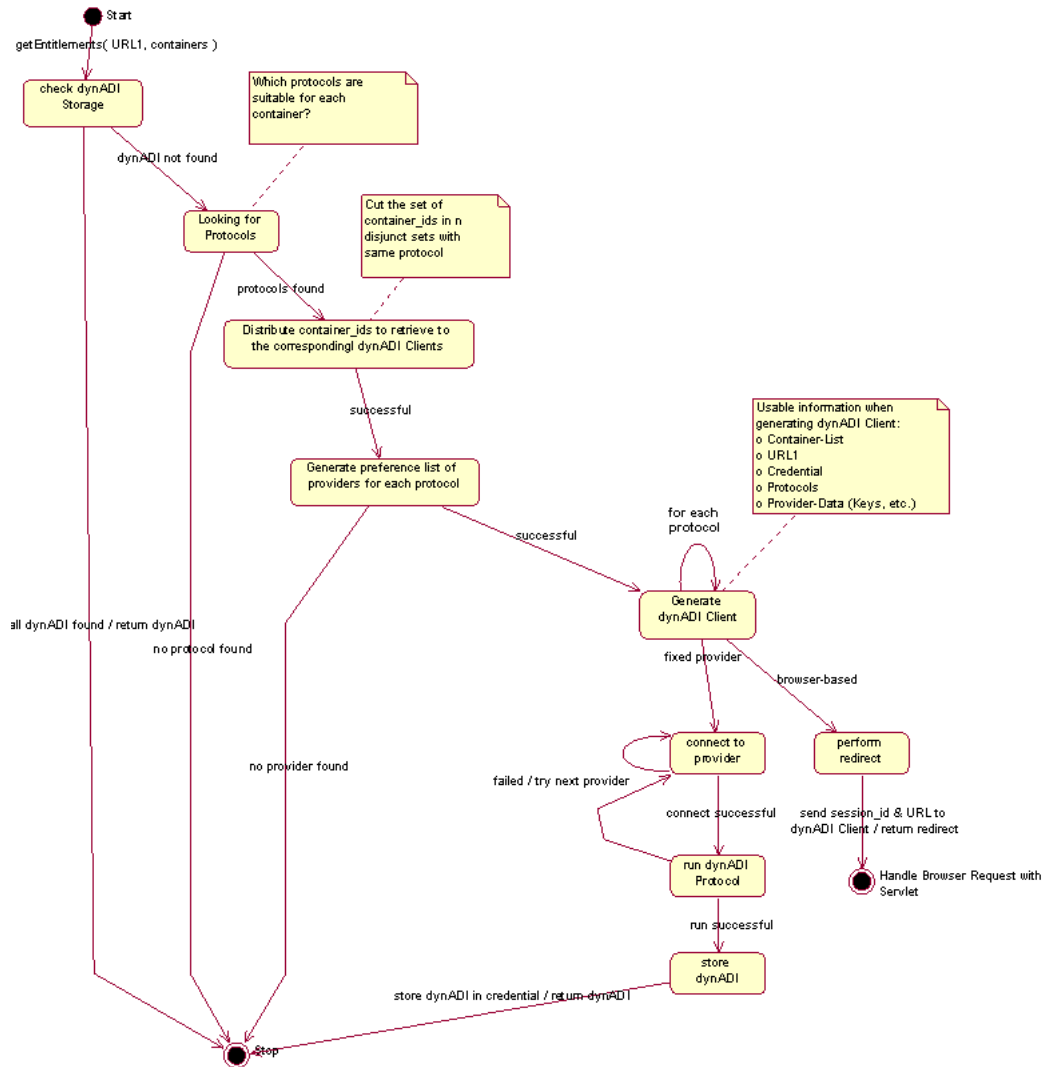


Figure A.3: State Chart of the DynADI Project

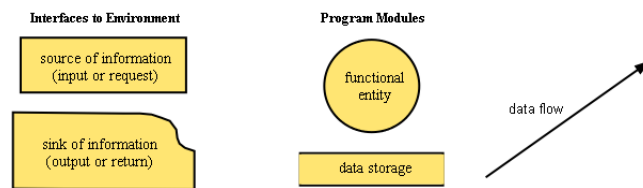


Figure A.4: Description of the Data Flow Diagram Notation

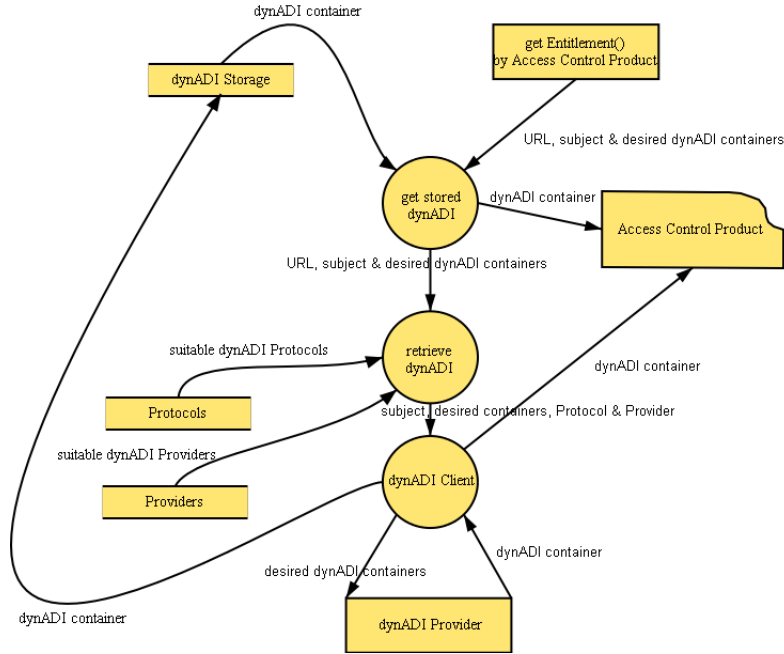


Figure A.5: Data Flow for a Fixed-provider Setting

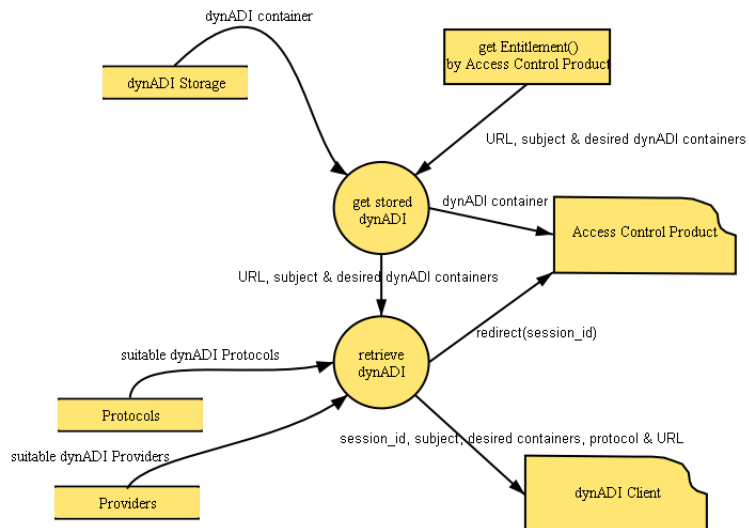


Figure A.6: Data Flow for a Browser-based Protocol

Container Descriptor Table (XML format) [RD240]

The table contains all information about the retrievable container types, the structure of the corresponding containers and the cost of the retrieval. We introduce the XML format of a container descriptor in Section A.5.1.

Key Storage File (Default Java key storage format) [RD300]

The DynAdiEntitlementService saves its keys in a KeyStorage provided by the Java Security API.

Exception Log File (Default jLog Format) [RD400]

The exception log file contains all exceptions thrown by the DynAdiEntitlementService and additional information about the recovery from the error.

Metering Log File (Default jLog Format) [RD420]

The DynAdiEntitlementService stores an entry for each successfully retrieved container in this log file. The log file is used to keep track of the retrieval cost.

Trace Log File (Default jLog Format) [RD440]

This log file contains a trace of the method calls of the DynAdiEntitlementService.

Protocol specific Log Files (Proprietary Format) [RD460]

The DynAdiProtocols may provide additional log information in proprietary formats i.e. with the SOAP requests and responses of the communication with the server.

A.5.1 XML Format of the Container Descriptors

We use the following XML scheme for exchanging the container descriptors:

```
<ContainerDescriptor>
  <container_type_id>id_data</container_type_id>
  <namespace_prefix>namespace_data</namespace_prefix>
  <cost>value_per_retrieval</cost>
  <protocol_id>id_data</protocol_id>

  <ContainerProperties>
    <property_attribute_name1>data1</property_attribute_name1>
    ... property-data ...
    <property_attribute_nameN>dataN</property_attribute_nameN>
  </ContainerProperties>

  <ContainerPayloadFormat>
    ... container-payload-format-data ...
  </ContainerPayloadFormat>
</ContainerDescriptor>
```

A.6 Product Performance

- No unusual time bounds.
- The response time of the DynAdiEntitlementService depends on the Round Trip Times (RTT) of the connections to the attribute providers and their response times.
- The time consumed by the infrastructure of the DynADI Entitlement Service is to be negligible compared with the RTT to the providers.

- If a `getEntitlements()` request queries `DynAdiContainers` retrieved from different providers, the response times are cumulative.

A.7 User Interface

No user interface. The DynADI plug-in is administrated with configuration files.

A.8 Quality Requirements

These are the quality requirements for the `DynAdiEntitlementService`. They are ordered by descending priority.

- **Reliability (A)**
In almost all program runs the service behaves according to the specification.
Note: To claim correctness we would require a formal verification of the `DynAdiEntitlementService`.
- **Modularity (B)**
The used primitives especially the protocols can be exchanged easily.
- **Robustness (C)**
The service is able to deal with malformed inputs from the access control product, the configuration files and the browser.

A.9 Test Considerations

The `DynAdiEntitlementService` is supposed to be fully tested with module tests. We use the technique of exemplary specification according to [ZS02] and [Bal96] to generate test cases for all classes of the `DynAdiEntitlementService` before their implementation. You design a test case for each method of a class which checks whether the method fulfills its specification. We describe this technique in Section 10.1.1.

Additionally in this section we show global test cases for the `DynAdiEntitlementService`. This consideration is base of the test harness for the service, which is provided by the function verification test crew of our customer.

We show with a \rightarrow , which functions are tested with the test case. If we state a generic term for several functions, we refer to the set of the functions below the the generic node in the function tree. We use the backslash character (`'\'`) to indicate that we exclude a function from that set. For instance \rightarrow [RF400]\[RF420] means: all `DynAdiStorage` functions without the function to remove DynADI from storage.

DynADI Clients [RT100-299] First we consider the test cases for the communication of the `DynAdiClients`.

- **Fixed-provider Setting [RT100]**
These test cases check the functions of the fixed-provider `DynAdiClients`.

- For each fixed provider:
Retrieve one DynAdiContainer from the provider **[RT110]**
→ [RF100]\[RF120,170], [RF310], [RF700]
This case tests the capability of each DynAdiClient to retrieve simple DynAdiContainers. We assume that the DynAdiStorage is empty.
- Retrieve several containers from different fixed providers using all protocols available **[RT120]**
→ [RF100]\[RF120,170], [RF310], [RF700]
Tests the functions of [RT110], but in one single getEntitlements() request. In addition to the functions above the correct management of different clients and protocols is tested.
- For each fixed provider supporting this:
Request a list of supported containers **[RT130]**
→ [RF100]\[RF130,180], [RF310]
tests the capability of the fixed-provider protocol to request a container list of a provider.
- For each fixed provider:
Ask for a DynAdiContainer type that is available in the container list but disabled at the DynAdiProvider. **[RT150]**
→ [RF130,180]
Tests whether the different DynAdiClients produce right error codes if the container type could not be retrieved.
- Request a container type that is not supported by the dynADI plug-in **[RT160]**
→ [RF670]
Tests whether a correct error code is produced if an unsupported container type is requested.
- Browser-based Setting **[RT200]**
Here we consider the test cases for the communication with a browser-based protocol.
 - For each browser-based protocol without need of storing dynADI in the DynAdiClients state: Request one container and follow the redirects **[RT210]**
→ [RF200]\[RF260], [RF310], [RF400]
These are simple browser-based requests which do not require to save the state of the DynAdiClient. The DynADI is stored in the DynAdiStorage between the redirects.
 - For each browser-based protocol with need of storing dynADI in the DynAdiClients state: Request one container and follow the redirects **[RT220]**
→ [RF200]\[RF260], [RF300]\[RF350]
As above, but this time the DynADI is stored in the state of the DynAdiClient. Therefore the client's state has to be saved after the first request and restored correctly in the second request.
 - Request several containers needing different browser-based protocols **[RT230]**
→ [RF200]\[RF260], [RF350]
This test focuses on the management of different browser-based protocols. All redirects have to be performed correctly.
 - For each browser-based protocol:
Request DynAdiContainer and use invalid values during the user interaction **[RT240]**
→ [RF200]\[RF260]

The DynAdiClients are supposed to return the correct error codes for the wrong inputs.

- Request a container with 'please wait' page [RT250]
→ [RF260]

Storing DynADIs [RT300-399] This set of test cases focuses on the functions of the DynAdiStorage and the capability to store DynADI in the credentials.

- Request a container with a browser-based protocol which saves the DynADI in DynAdiStorage. [RT310]
→ [RF400]\[RF420,450]
The browser-based protocol will save the DynADI after the redirect to the resource URL in the DynAdiStorage. The DynADI is supposed to be read from the storage when the browser requests the URL a second time.
- Request a container with a browser-based protocol which saves the DynADI in the credential. [RT320]
→ [RF450]\[RF470]
As above, but this time the DynADI is saved in the credential.
- For each browser-based protocol storing DynADI in DynAdiStorage:
Request a container await its timeout before following the redirect to the resource URL. [RT330]
→ [RF400]\[RF450]
This test case focuses on the correct handling of timeouts of the containers. The container has to be removed from the DynAdiStorage.
- For each browser-based protocol storing DynADI in credential:
Request a container await its timeout before following the redirect to the resource URL. [RT340]
→ [RF450]
This test case focuses on the correct handling of timeouts of the containers. The container has to be removed from the credential.
- For each browser-based protocol storing the DynADI in the state of the DynAdiClient:
Request a container await its timeout before following the redirect to the resource URL. [RT350]
→ [RF320,330]
This test case focuses on the correct timeout handling in the functions for storing the state of a DynAdiClient. The DynAdiClient is supposed to remove the DynADI from its state.

Manage protocol-specific data [RT400-499]

Appendix B

Glossary

B.1 Terminology

Access Control The protection of resources against unauthorized access.

Access Control List (ACL) Format for specification of an Access Matrix. The Access Control List is attached to a resource. It contains users, groups and permissions that they have on the resource. See also [Pfi00, SS94].

Access Matrix Is a set $ACC \subseteq S \times O \times R$, where S is a set of subjects, O a set of objects and R a set of rights. The subjects are active entities of a system, the objects passive entities and the rights possible ways a subject can access a object. One entry (s, o, r) of an Access Matrix models that a subject s has the right r on object o . [Pfi00, SS94]

Access Control Product managing access control and deciding authorization queries. It has the capability to evaluate Boolean Rules on the Access Decision Information (ADI). We also use the word Access Control Software.

Access Decision Information (ADI) We call the set of information taken into account to decide authorization queries Access Decision Information (ADI). In general these are Credentials, Entitlements, Protected Object Policy (POP) entries and information about a HTTP request. See Definition 1 on Page 12.

Adversary A dishonest party that tries to break a protocol.

Application Server A web server that is specialized in hosting active web applications like Servlets, Java Server Pages, etc.

Attribute A distinct characteristic of an object or a subject. A subject's attribute could for instance be the age, credit line, service level, country of birth, etc. [SAML02b, SAML02c].

Attribute Provision The process of retrieving dynamic ADI from a DynAdiProvider.

Back-end Servers These are servers with sensitive or valuable data. A Web-Access Control Product secures them.

Boolean Rule A conditional over boolean statements about dynamic attributes. With given attribute values the each statement and the whole rule can be evaluated to the boolean values true or false.

Browser In general a Browser is a tool used to view and surf the World Wide Web. In our access control scenario a browser tries to access web pages on one or more back-end servers secured by an Access Control Product.

Browser-based A type of DynAdiProtocol. The protocol interacts with the user's browser in order to get the required ADI. The easiest case is that the user is directly asked to enter ADI.

Browser-based Attribute Exchange (BBAE) A protocol for browser-based attribute provision. It allows anonymity and is designed to be privacy friendly [PW02b, PW02a]. See Section 2.2.1 on Page 16.

Capability List Format for specification of an Access Matrix. It specifies the capabilities of a user or group. It contains references to resources and the permissions on that resource. See also [Pf00, SS94].

Context-based Access Control Access Control based on attributes of a user or the application-dependent context. Usually the attributes used there are dynamic.

Credential A data-structure that contains data about a subject's identity, group memberships and attributes.

Design Pattern An element of reusable software that solves a certain problem on an abstract level [GHJV95]. See Section 6.4 on Page 59.

Destination Site A Destination Site consumes identity or attribute assertions of Source Sites. Destination Sites are for instance e-mail portals or e-commerce or e-banking server.

DynADI plug-in other name for the DynAdiEntitlementService.

DynAdiClient A client module of the DynAdiEntitlementService that runs a special DynAdiProtocol in order to communicate with one DynAdiProvider.

DynAdiEntitlementService is used to retrieve missing ADI on request. It encapsulates different entitlement provision protocols. This is the central component of our project. For short we also call this service DynADI plug-in. The DynAdiEntitlementService is associated with two other components: the DynAdiStorage and the DynAdiClient. We introduce its design in Section 9.1 on Page 76.

DynAdiProtocol A particular entitlement provision protocol of a given vendor. Such a protocol is used to communicate with a DynAdiProvider. We distinguish different types of DynAdiProtocols: fixed-provider and browser-based. (See also Federated Design)

DynAdiProvider A service that can provide ADI. Such a provider can be web-based or not. It may be on a local web server or not.

DynAdiStorage This component caches DynADI temporarily. The duration is specified by the protocol used. The DynADI may also be stored in the state of a DynAdiClient.

Dynamic Access Decision Information (DynADI) Access Control Information (ADI) that is retrieved dynamically. In most cases this information has a short validity time and has to be retrieved on demand.

Entitlement An attribute assertion of a third party issued to an Access Control Product.

Extensible Markup Language (XML) Extensible Markup Language describes a class of data objects called XML documents. These documents contain so called XML Elements. The structure of these XML documents is described in so called XML Schemas.

Federated Design We call a design federated, if it allows both DynAdiProtocol types: fixed-provider and *browser-based*.

Fixed-provider A type of DynAdiProtocol. The DynAdiEntitlementService directly contacts the server of a DynAdiProvider.

Hypertext Transfer Protocol (HTTP) This request-response protocol is the base protocol of the World Wide Web. If a browser accesses a web page, the corresponding data is requested with HTTP Requests from the server and delivered in the corresponding HTTP Responses. [RFC1945, RFC2616]

HTTP Request A browser sends a HTTP Request to a web server to retrieve data from it. The request specifies the data with a URL. The server answers the request normally within the same connection with a HTTP Response.

HTTP Response A servers send requested data in a HTTP Response back to the browser. The response can also contain error messages or Redirects to other URLs.

Identifier A representation (for example, a string) mapped to a system entity that uniquely refers to it. [SAML02c]

jUnit A Java framework for automated module tests. See Section 10.1.2 (Page 96).

Login Synonym for Sign-on.

Nonce A nonce is a cryptographically secure random number. It is used as freshness measure in protocol design and provides replay prevention and actuality. Normally a receiver of a message chooses the nonce and sends it to the sender. The sender signs the message together with the nonce.

Organization for the Advancement of Structured Information Standards (OASIS)

According to its own mission statement the OASIS is a not-for-profit, global consortium that drives the development, convergence and adoption of e-business standards. It developed the Security Assertion Markup Language (SAML). Further information about this consortium can be obtained from <http://www.oasis-open.org/>. On <http://www.oasis-open.org/about/index.shtml> you can find Information about the member organizations of OASIS.

Protected Object Policy (POP) A security policy that is specific for a certain resource and is attached to it.

Redirect A HTTP response type which states that further action must be taken to fulfill the request. The redirect we deal with has the code 302 and states that the requested resource resides temporarily under a different URI. On receiving such a HTTP Response a browser will try to retrieve the new URI. [RFC2616].

Request for Comments (RFC) A proposal of a technical description for the internet. A database of RFC documents can be found at <http://www.rfc-editor.org/rfcsearch.html>.

Resource A resource in our sense can be a piece of data or a service provided by a system.

- SAML Artifact** A small piece of data that refers non-ambiguously to a SAML Assertion. [SAML02b]
- SAML Assertion** A piece of data that formalizes an assertion about a subject's identity or attributes. It regards either an act of authentication performed on a subject, attribute information about the subject, or authorization permissions applying to the subject with respect to a specified resource. [SAML02c, SAML02a] See Section 2.1 on Page 13.
- SAML Binding** An instance of mapping SAML request-response message exchanges into a specific communication protocol [SAML02c, SAML02b].
- SAML Profile** A set of rules describing how to embed SAML assertions into and extract them from a framework or protocol [SAML02c, SAML02b].
- SAML Searchpart** Searchpart of a URL that contains one or more SAML artifacts and a description of a target. It is defined in Definition 5 on Page 28. [SAML02b]
- Secure Hypertext Transfer Protocol (HTTPS)** The Secure HTTP protocol utilizes secure communication channels generated with the protocols Secure Socket Layer (SSL) and Transport Layer Security (TLS) to transfer HTTP messages. HTTPS provides authentication with client and server certificates. [RFC2818]
- Secure Socket Layer (SSL)** SSL is the predecessor of TLS.
- Security Assertion Markup Language (SAML)** A message and protocol standard of the OASIS consortium. It contains a set of specifications describing security assertions that are encoded in XML [SAML02a], profiles for attaching the assertions to various protocols and frameworks, and bindings of this protocol to various transfer protocols [SAML02b, SAML02c]. See Section 2.1 on Page 13.
- Session** A lasting interaction between system entities, often involving a user, typified by the maintenance of some state of the interaction for the duration of the interaction. [SAML02c]
- Sign-on** The process of authentication to a system.
- Simple Object Access Protocol (SOAP)** SOAP is a XML-based protocol for exchange of information in a decentralized, distributed environment. [SOAP00]
- Single Sign-on** A user logs in at a Source Site and authenticates to that site. The Source Site confirms an identity of the user to other sites (Destination Sites). The user only needs to authenticate to the Source Site and does not need to identify at each Destination Site.
- Site** An administrative domain in geographical or DNS name sense.
- Source Site** A Source Site is a site that provides identity assertions to other parties (Destination sites).
- Subject** A system entity whose identity can be authenticated in the context of Access Control. [SAML02c]
- Transport Layer Security (TLS)** A protocol that generates secure point-to-point connections. The connections provide confidentiality and integrity as well as freshness and robustness measures. Unilateral or bilateral authentication is possible. The successor of SSL. [RFC2246]

Uniform Resource Identifier (URI) A compact string of characters for identifying an abstract or physical resource. URIs are the universal addressing mechanism for resources on the World Wide Web. The URLs are a subset of the URIs. [SAML02c]

Uniform Resource Locator (URL) Uniform Resource Locators (URLs) are a subset of URIs that use an addressing scheme tied to the resources primary access mechanism, for example, their network “location”. [SAML02c]

Coordinated Universal Time (UTC) A format for expressing date and time values, which also used for the W3C XML Schema Data-types.

Web Other name for the World Wide Web.

Web-Access Enforcement Product This system shields and protects resources on the back-end servers while asking an access control software for authorization. It enforces the decisions taken by an Access Control Product.

Web Browser See Browser.

Web Service A Web Service is a self-describing, self-contained, modular application. It provides some functionality to other applications through an Internet connection.

Web Service Description Language (WSDL) XML language for describing Web Services.

World Wide Web (WWW) The worldwide network of web pages in the internet. The web pages are retrieved with the application protocols HTTP and HTTPS.

XML Element An XML data structure that is hierarchically arranged among other such structures in an XML document and is indicated by either a start-tag and end-tag or an empty tag. [SAML02c]

XML Schema The format developed by the World Wide Web Consortium (W3C) for describing rules for a markup language to be used in a set of XML documents. [SAML02c]

B.2 Abbreviations

ACL Access Control List

ADI Access Decision Information

BBAE Browser-based Attribute Exchange

DynADI Dynamic Access Decision Information

HTTP Hypertext Transfer Protocol

HTTPS Secure Hypertext Transfer Protocol

JNI Java Native Interface

OASIS Organization for the Advancement of Structured Information Standards

POP Protected Object Policy

RFC Request for Comments

SAML Security Assertion Markup Language
SAML SSO SAML Single Sign-on Browser/Artifact (Profile)
SAMLart SAML artifact
SAMLsp SAML searchpart
SAMLspEnh Enhanced SAML searchpart
SOAP Simple Object Access Protocol
SSL Secure Socket Layer
SSO Single Sign-on
TLS Transport Layer Security
WS Web Service
WSDL Web Service Description Language
WSTK Web Services Toolkit
WWW World Wide Web
UTC Coordinated Universal Time
URI Uniform Resource Identifier
URL Uniform Resource Locator
XML Extensible Markup Language

Bibliography

- [All02] Ant Allan. IBM/Tivoli software tivoli policy director (TPD) extranet access management (EAM) product, 2002.
- [BW98] Philip Bishop and Nigel Warren. Java tip 78: Recycle broken objects in resource pools, 1998. Available from World Wide Web: http://www.javaworld.com/javatips/jw-javatip78_p.html.
- [Bal96] Helmut Balzert. *Lehrbuch der Software-Technik I - Software-Entwicklung*. Spektrum Akademischer Verlag, 1996.
- [Bel97] Doug Bell. Make java fast: Optimize!, 1997. Available from World Wide Web: <http://www.javaworld.com/javaworld/jw-04-1997/jw-04-optimize.html>.
- [Bha01] Bhavin Bharat Bhansali. Man-in-the-middle attack - a brief, February 2001. Available from World Wide Web: <http://www.sans.org/rr/threats/middle.htm>.
- [Blo01] Joshua Bloch. *Effective Java Programming Language Guide*. Addison Wesley Professional, 1st edition, 2001.
- [CH02] Jan Camenisch and Els Van Herreweghen. Design and implementation of the Idemix anonymous credential system. In *9th ACM Conference on Computer and Communications Security*, volume RZ 3419. ACM Press, November 2002. Available from World Wide Web: <http://www.zurich.ibm.com/security/publications/2002/camher02b.pdf>.
- [Cam01] Camelot. Differentiating between access control terms, 2001. Available from World Wide Web: http://www.windowsecurity.com/uplarticle/2/Access_Control_WP.pdf.
- [Cer02] Ethan Cerami. *Web Services Essentials*. O'Reilly and Associates, Inc., first edition, February 2002.
- [DM02] Paras Dave and Nathan Moussa. TCP connection hijacking, 2002. Available from World Wide Web: <http://cs.baylor.edu/~donahoo/NIUNet/hijack.html>.
- [Dav98] Thomas E. Davis. Build your own ObjectPool in java to boost app speed, 1998. Available from World Wide Web: http://www.javaworld.com/javaworld/jw-06-1998/jw-06-object-pool_p.html.
- [Ele02] The Mind Electric. *GLUE 2.3 User Guide*. The Mind Electric, 2002. Available from World Wide Web: <http://www.theminelectric.com/glue/index.html>.
- [GHJV95] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns*. Addison-Wesley, 1995. Elements of reusable object-oriented software.

- [GJM91] Ghezzi, Jazayeri, and Mandrioli. *Fundamentals of Software Engineering*. Prentice Hall, 1991.
- [Har97] Jonathan Hardwick. Java optimization, 1997. Available from World Wide Web: <http://www-2.cs.cmu.edu/jch/java/optimization.html>.
- [KR00] David P. Kormann and Aviel D. Rubin. Risks of the passport single signon protocol. *Computer Networks*, 33:51–58, 2000. Available from World Wide Web: <http://avirubin.com/passport.htm>.
- [Kah01] Bernd Kahlbrandt. Wasserfall-modell, December 2001.
- [Kar01] Günter Karjoth. The authorization service of tivoli policy director. *17th Annual Computer Security Applications Conference (ACSAC 2001)*, 2001.
- [Kre01] Heather Kreger. Web services conceptual architecture (WSPA 1.0), May 2001. Available from World Wide Web: <http://www.ibm.com/developerworks/webservices>.
- [LIB03a] Jeff Hodges and Tom Wason. Liberty architecture overview, 2003. Available from World Wide Web: <http://www.projectliberty.org/>, <http://www.projectliberty.org/specs/main.html>.
- [LIB03b] Jason Rouault and Tom Wason. Liberty bindings and profiles specification, 2003. Available from World Wide Web: <http://www.projectliberty.org/>, <http://www.projectliberty.org/specs/main.html>.
- [LIB03c] John D. Beatty and John Kemp. Liberty protocols and schema specification, 2003. Available from World Wide Web: <http://www.projectliberty.org/>, <http://www.projectliberty.org/specs/main.html>.
- [Mea96] Meadows. Analyzing the needham-schroeder public-key protocol: A comparison of two approaches. In *ESORICS: European Symposium on Research in Computer Security*. LNCS, Springer-Verlag, 1996. Available from World Wide Web: citeseer.nj.nec.com/meadows96analyzing.html.
- [Mic02a] Microsoft. .net passport overview, 2002. Available from World Wide Web: <http://www.microsoft.com/netservices/passport/default.asp>.
- [Mic02b] Microsoft. .net passport review guide, 2002. Available from World Wide Web: <http://www.microsoft.com/netservices/passport/passport.asp>.
- [NHMT98] U. Nitsche, R. Holbein, O. Morger, and S. Teufel. Realization of a context-dependent access control mechanism on a commercial platform. In *Proceedings of IFIP/SEC 1998*, Vienna (Austria) and Budapest (Hungary), 1998. Chapman & Hall. Available from World Wide Web: citeseer.nj.nec.com/nitsche98realization.html.
- [PW02a] Birgit Pfitzmann and Michael Waidner. BBAE – a general protocol for browser-based attribute exchange. Research report RZ 3455 (# 93800), IBM Research, June 2002. Available from World Wide Web: <http://www.zurich.ibm.com/security/publications/2002.html>.
- [PW02b] Birgit Pfitzmann and Michael Waidner. Privacy in browser-based attribute exchange. *ACM Workshop on Privacy in the Electronic Society*, Washington, November 2002.
- [PW02c] Birgit Pfitzmann and Michael Waidner. Token-based web single signon with enabled clients. 2002.

- [Pfi00] Birgit Pfitzmann. *Practical Security*. Saarland University, Saarbrücken, 2000. Available from World Wide Web: <http://www-krypt.cs.uni-sb.de>.
- [Pfi01] Birgit Pfitzmann. *Kryptographie*. Saarland University, Saarbrücken, June 2001. Available from World Wide Web: <http://www-krypt.cs.uni-sb.de>.
- [Pfi99] Birgit Pfitzmann. *Higher Cryptographic Protocols*. Saarland University, Saarbrücken, 1999. Available from World Wide Web: <http://www-krypt.cs.uni-sb.de>.
- [RFC1510] John Kohl and B. Clifford Neuman. RFC 1510: The kerberos network authentication service (version 5), September 1993. Available from World Wide Web: <ftp://ftp.rfc-editor.org/in-notes/rfc1510.txt>. Status: Standards track.
- [RFC1945] T. Berners-Lee, R. Fielding, and H. Frystyk. RFC 1945: Hypertext Transfer Protocol — HTTP/1.0, May 1996. Available from World Wide Web: <ftp://ftp.internic.net/rfc/rfc1945.txt>, <ftp://ftp.math.utah.edu/pub/rfc/rfc1945.txt>. Status: Informational.
- [RFC2119] S. Bradner. RFC 2119: Key words for use in RFCs to indicate requirement levels, March 1997. Available from World Wide Web: <ftp://ftp.rfc-editor.org/in-notes/rfc2119.txt>. Status: Best current practice.
- [RFC2145] J. C. Mogul, R. Fielding, J. Gettys, and H. Frystyk. RFC 2145: Use and interpretation of HTTP version numbers, May 1997. Available from World Wide Web: <ftp://ftp.rfc-editor.org/in-notes/rfc2145.txt>. Status: Informational.
- [RFC2246] T. Dierks and C. Allen. RFC 2246: The TLS protocol, January 1999. Available from World Wide Web: <ftp://ftp.rfc-editor.org/in-notes/rfc2246.txt>. Status: Standards Track.
- [RFC2616] R. Fielding, J. Gettys, J. C. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. RFC 2616: Hypertext transfer protocol – HTTP/1.1, June 1999. Available from World Wide Web: <ftp://ftp.rfc-editor.org/in-notes/rfc2616.txt>. Status: Standards Track.
- [RFC2818] E. Rescorla. RFC 2818: HTTP over TLS, May 2000. Available from World Wide Web: <ftp://ftp.rfc-editor.org/in-notes/rfc2818.txt>. Status: Informational.
- [SAML02a] Phillip Hallam-Baker et al. Assertions and protocol for the OASIS security assertion markup language (SAML), 2002. Available from World Wide Web: <http://www.oasis-open.org/committees/security/>.
- [SAML02b] P. Mishra et al. Bindings and profiles for the OASIS security assertion markup language (SAML), 2002. Available from World Wide Web: <http://www.oasis-open.org/committees/security/>.
- [SAML02c] Jeff Hodges and Eve Maler. Glossary for the OASIS security assertion markup language (SAML), May 2002. Available from World Wide Web: <http://www.oasis-open.org/committees/security/>.
- [SAML02d] OASIS. SAML assertion schema cs-sstc-schema-assertion-01.xsd, 2002. Available from World Wide Web: <http://www.oasis-open.org/committees/security/>.
- [SHIB02] Scott Cantor and Marlena Erdos. Shibboleth-architecture draft v05, May 2002. Available from World Wide Web: <http://shibboleth.internet2.edu/docs/draft-internet2-shibboleth-arch-v05.pdf>.

-
- [SOAP00] Don Box, David Ehnebuske, Gopal Kakivaya, Andrew Layman, Noah Mendelsohn, Henrik Frystyk Nielsen, Satish Thatte, and Dave Winer. Simple object access protocol (SOAP) 1.1, May 2000. Available from World Wide Web: <http://www.w3.org/TR/SOAP/>.
- [SS94] Ravi S. Sandhu and Pierrangela Samarati. Access control: Principles and practice. *IEEE Communications Magazine*, 32(9):40–48, 1994.
- [Sle01] Marc Slemko. Microsoft passport to trouble, 2001. Available from World Wide Web: <http://alive.znep.com/marcs/passport/>.
- [Sne02] James Snell. Implementing web services, 2002. Available from World Wide Web: <http://www.ibm.com/developerworks/webservices>.
- [ZS02] Andreas Zeller and Gregor Snetling. *Softwaretechnik I*. Saarland University, 2002. Available from World Wide Web: <http://st.cs.uni-sb.de/>.