idFS Design Documentation

Version: 2.3
Date: 19.03.2008
Author: Johannes Meinecke

Contents

1 Architecture

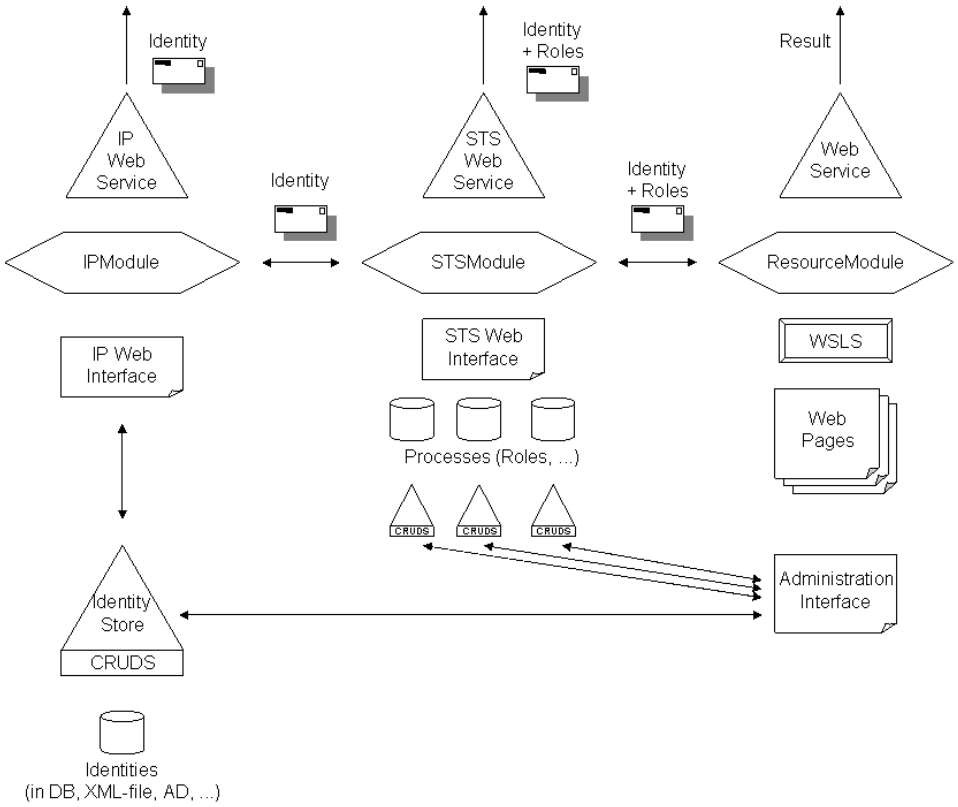

Figure 1: Implementation architecture

2 Account Life Cycle

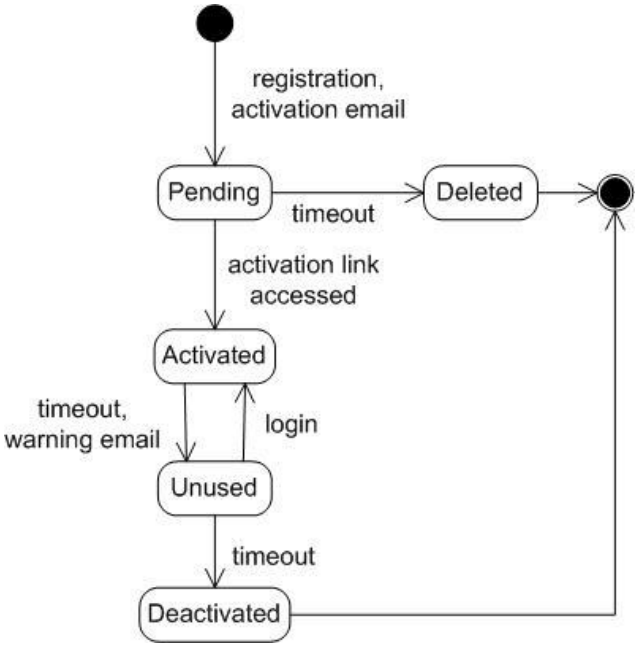

Figure 2: Life cycle of a user account

3 Protocols

3.1 PRP Protocol

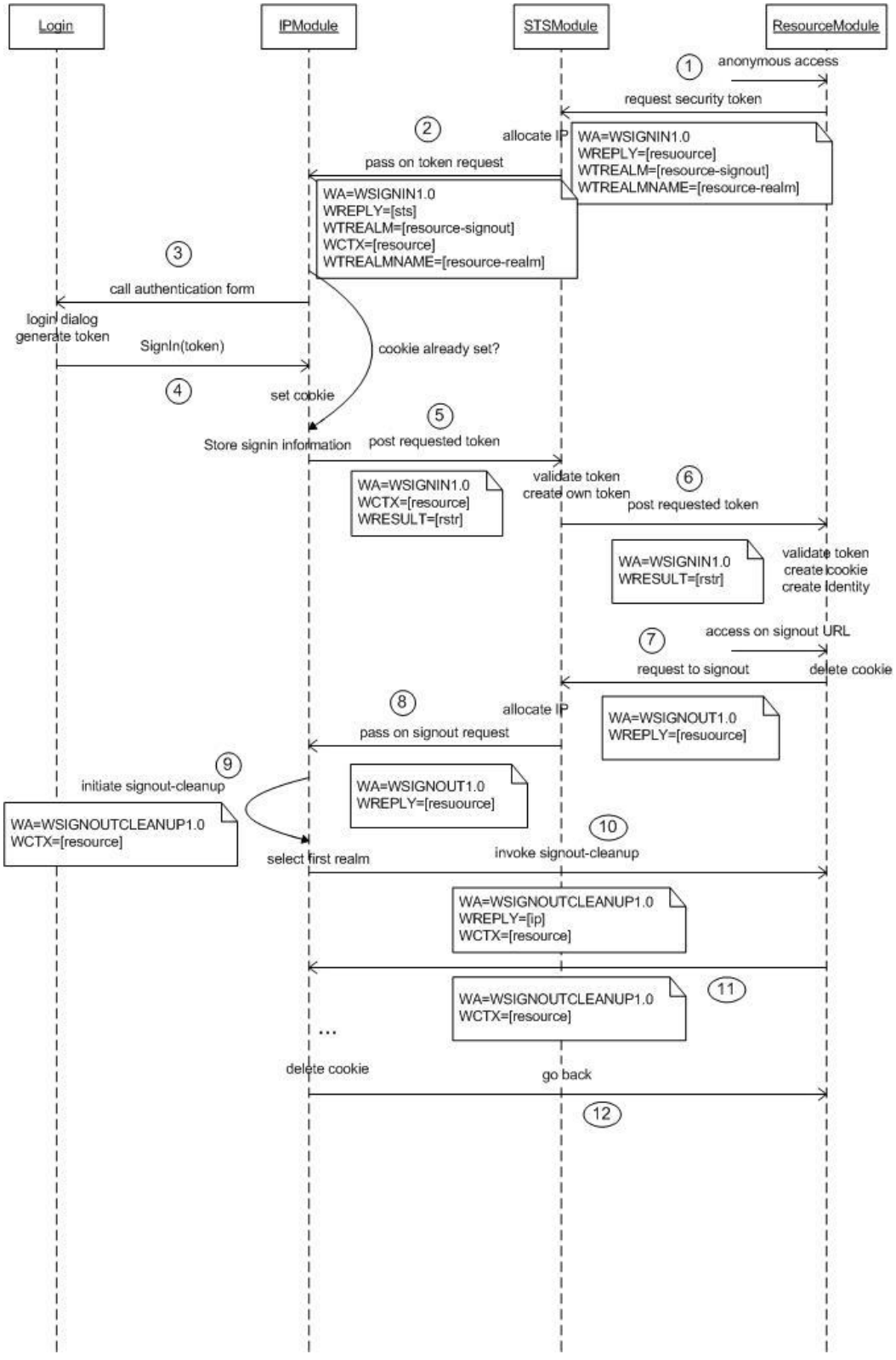

Figure 3: Sequence diagram for the Passive Requestor Profile

3.2 Classes in PassiveRequestorProfile Namespace

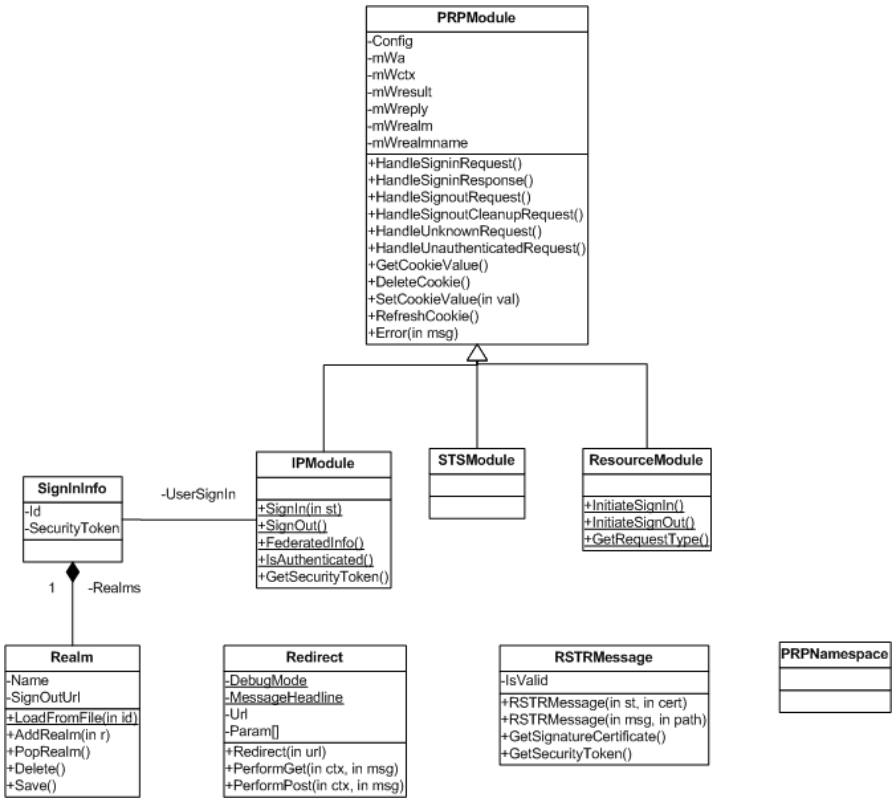

Figure 4: Namespace WSLS.Federation.PassiveRequestorProfile.PassiveRequestorProfile

- **PRPModule**: Base class for HTTP modules that handles PRP parameters, PRP configuration, authentication cookies etc.
- **IPModule**: HTTP module for IPs.
- **STSModule**: HTTP module for STS.
- **ResourceModule**: HTTP module for resources.
- **SignInInfo**: Represents session information about a user logged in at an IP.
- **Realm**: Represents information about a security realm (of an application) a user is signed into.
- **Redirect**: Auxiliary class for simple HTTP redirects with support for parameters and debug messages.
- **RSTRMessage**: Represents a signed SOAP message for transporting security tokens.
- **PRPNamespace**: Contains constants for the PRP namespace.

3.3 ARP Protocol

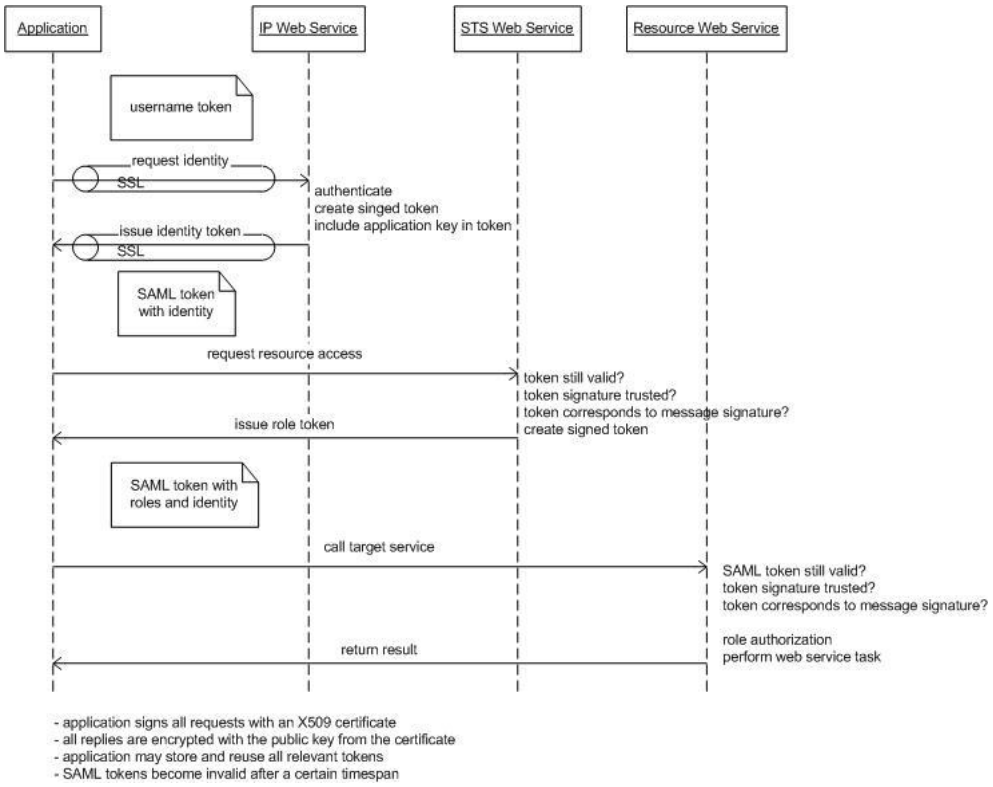

Figure 5: Sequence diagram for the Active Requestor Profile

4 Identity Provider

4.1 Identity Attributes

Attribute	Type	Comment	Meta	auto	token	in Active Directory
Identifier	String		yes		yes	objectguid
LoginName	String				yes	samaccountname
Password	String					[password]
LastLoginTime	String					comment
IsPending	Boolean					comment
IsActive	Boolean					-
ActivationID	String	→pending users				comment
RegistrationTime	DateTime	→DC: Date	yes			whencreated
LastModified	DateTime		yes			whenchanged
Creator	String	=Identifier	yes			comment
IsModifiable	Boolean			yes	yes	-
IsUnused	Boolean					comment
AuthQuestion	String					comment
AuthAnswer	String					comment
NewEmailAddress	String					comment
NewEmailID	String	ID for confirmation				comment
Salutation	String	"Herr", "Mrs.", ...				comment
AcademicTitle	String	"Dr.", ...				personaltitle
FirstName	String				yes	givenname
LastName	String				yes	sn
Title	String	display name	yes	yes		-
Type	String	e.g. "Student,DA"	yes			comment
Language	String		yes			preferredlanguage
StreetAddress	String					streetaddress
City	String					l
PostalCode	String					postalcode
State	String					st
Country	String	e.g. "DE"				c
JobPosition	String					title
Department	String					department
Company	String					company
TelPrivate	String					homephone
TelOffice	String					telephonenumber
TelMobile	String					mobile
Fax	String					facsimiletelephonenumber
EmailAddress	String	used for init. mail			yes	mail
EmailAdress2	String					othermailbox
HomepageUrl	String					wwwhomepage
StudentNumber	String	→Matrikelnummer				employeenumber
StudentSubject	String	e.g. "Informatik"				employeetype
Source	String	URL of IP	yes	yes	yes	-

Figure 6: Attributes that define an identity

4.2 Interface of the Identity Store Service

```
// Returns the identity of the user with the supplied credentials or null.
public WSLS.Adapters.GTS.ContentObject Authenticate(string loginName,
   string password)

// Registers the specified identity and returns an identity object
   containing the activation id.
public WSLS.Adapters.GTS.ContentObject
   Register(WSLS.Adapters.GTS.ContentObject identity)

// Activates a pending object with the specified activation id.
public bool Activate(string activationID)

// Deletes old pending accounts, marks unused accounts (+returns them) and
   deactivates old unused accounts.
public WSLS.Adapters.GTS.ContentObject[] CleanUp()

// Updates an identity with the specified data.
public void Modify(WSLS.Adapters.GTS.ContentObject identity)

// Initiates a change of the email address and returns an ID that can be
   used to confirm the change.
public string ChangeEmailAddress(string identifier, string newAddress)

// Confirms a changed email address with the specified ID. Returns true, if
   successful.
public bool ConfirmChangedEmail(string newEmailID)

// Changes the password of an identity to the specified value.
public void ChangePassword(string identifier, string password)

//  Returns the identity of the user with the supplied soft credentials or
   null.
public WSLS.Adapters.GTS.ContentObject AuthenticateWithQuestion(string
   loginName, string question, string answer)
```

5 Security Token Service

5.1 Tables at the Security Token Service

Role	
Identifier	key
Name	same as Title
RoleId	same as Identifier

AutorizationRule	
Identifier	key
Condition	string,i.e. "name='meinecke'"
IPUrl	normalized, can also be null
RoleId	id of role to be granted (Role.Identifier)
Index	priority value of the rule

RoleOwnership	
Identifier	key
IdentityId	can also be null for templates (Identity.Identifier)
RoleId	id of owned role (Role.Identifier)
Date	time of issuance
ValidFrom	
ValidUntil	
ValidNr	max. issue counter
IssueCounter	number of sign-ins
TokenNr	e.g. position on registration list
TokenFrequency	i.e. how often is the token owned
Type	e.g. “NumberedToken”

ActivationCode	
Identifier	key
OwnershipId	id of ownership template (RoleOwnership.Identifier)
ValidFrom	
ValidUntil	
ValidNr	max. issue counter
IssueCounter	
Code	to be entered
Message	displayed when issued

IPAllocationRule	
Identifier	key
Condition	string,i.e. "agent='ie'"
IPUrl	url of Identity Provider
Index	priority value of the rule

Figure 7: Tables / Data Web services at the Security Token Service

5.2 Authorization Mechanisms at the STS

The STS performs two main tasks:

- to **allocate the correct IP** to the anonymous requestors
- to **calculate a set of roles** to be issued to the requestor after he has been authenticated

The role calculation can be seen as a **black box**:

- **input**: a security token describing an identity
- **output**: a set of roles (from the point of view of the application, one particular role can either be present once or not at all; any other information like e.g. the token frequency is not passed on during the authorization process)

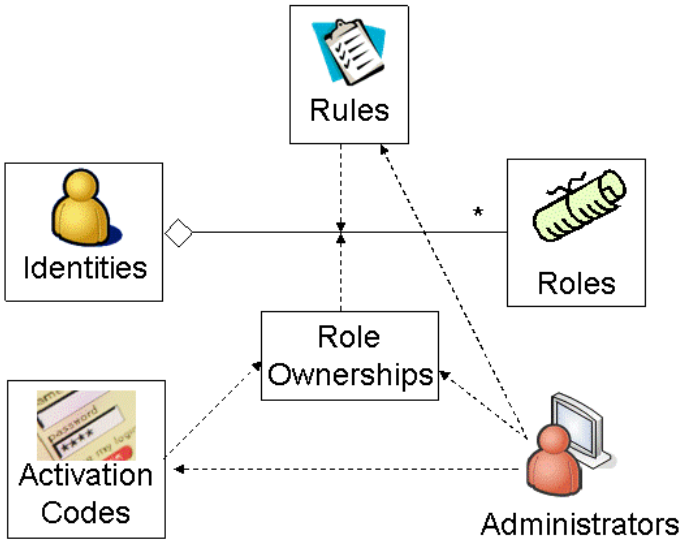

Figure 8: Basic authorization mechanism

- The STS issues roles to identities via rules and via role ownerships,
- Rules can be used when the identities are unknown in advance, e.g. in federated scenarios to issue roles to identities from foreign identity providers.
- Role Ownerships can issue roles dynamically, e.g. time-restricted, frequency-restricted, …
- Activation codes define possible ways for the users to acquire role ownerships.

5.2.1 IP Allocation Rules

- the rules are **applied in the order** “sorted by index”.
- rules at the top **can be overwritten** by rules at the bottom
- the conditions are written in **ADO.NET query syntax**
- the variables to be compared are taken from `HttpContext.Current.Request.Params`; they include **HTTP header parameters** as well as **parameters form the URL query string**
- the first rule should be a **default rule** with the condition “true”
- for further details cf. **AllocationRules.doc**

5.2.2 Authorization Rules

- the conditions are written in **ADO.NET query syntax**

- the variables to be compared are taken the from set of **attributes in the identity token**
- a particular role is only issued to a requestor **once at most** and does not have any other properties (as in the case of owned roles)
- as a built-in rule, the **AuthenticatedUser** role is always issued

5.2.3 Role Ownerships

- a role can be owned in several different ways:
  - **permanentRole**: no restrictions
  - **temporaryRole**: only valid between `validFrom` and `ValidUntil`
  - **nTimeRole**: can only be issued `ValidNr` times by the STS (i.e. restricted number of logins)
  - **numberedRole**: the role ownership is characterized by a number
  - **accumulatingRole**: the role can be owned more than once
- roles are only issued, if the **ownership** is currently **valid**
- **obsolete** ownerships are **not deleted** automatically
- a **counter** stores the number of logins, at which the role has been issued
- the same role can be **owned more then once** in different modi

5.2.4 Activation Codes

- codes can be used to activate **different types of role ownerships** (see above)
- entered codes must be **known** and **currently valid** (time limit, frequency limit)
- **obsolete** codes are **not deleted** automatically
- they can **only** be used **once** for a single identity
- a **counter** stores the number of times the code has been issued
- **numberedRole**-ownerships automatically receive ascending numbers

6 Resource

6.1 Protection Mechanism for CRUDS Web Services

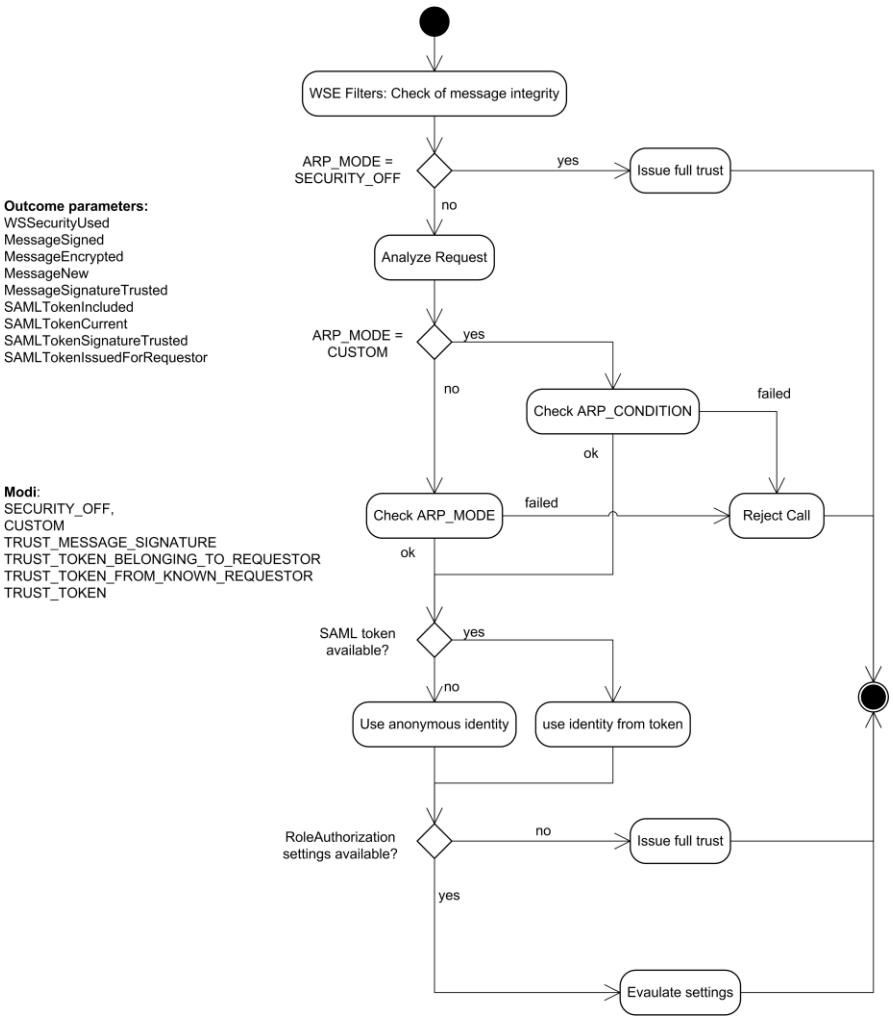

Figure 9: Authorization Mechanism for requests to CRUDS services

	Create	Read	Update	Delete	Search
SearchAndRead					X
UpdateAsXml			X		
SupportedQueryTypes					X
CreateAsXml	X				
Create	X				
Search					X
ReadApplicationLog	X	X	X	X	X
Delete				X	
ReadGTSIdentifiers		X			
ReadRSS					
ReadAsXml		X			
Update			X		
SearchAndReadXml					X
GetServiceCard		X			
Read		X			
SupportedSearchColoumns					X

Figure 10: Required permissions for accessing CRUDS web methods

6.2 PRP Interface to the Resource

There are 4 possible cases for the resource site, dependent on the value of `((AccountInfo)(System.Web.HttpContext.Current.User)).PRPStatus`:

- **`anonymous`:** This can only happen if the configuration setting `InterceptUnauthenticatedRequest` has been set to `false`. The resource site can determine itself, whether the user should be redirected to the STS for authentication. The method `ResourceModule.InitiateSignIn()` can be used for that purpose.
- **`signoutcleanup`:** This can only happen if the configuration setting `InterceptSignOutCleanUp` has been set to `false`. The resource site can take any necessary steps to sign out the user locally. Afterwards, `ResourceModule.FinishSignOutCleanUp()` should be called to redirect back to the IP.
- **`authenticated`:** In this case the user has already been authenticated earlier. The resource site might have to consult a cookie of its own to identify the user. The method `ResourceModule.InitiateSignOut()` can be used to sign out the user at the IP.
- **`signin`:** Directly after the user has signed in, the resource site is accessed for the first time. The generated identity contains the user data (only once). The resource site might want to generate a cookie of its own to remember the user.

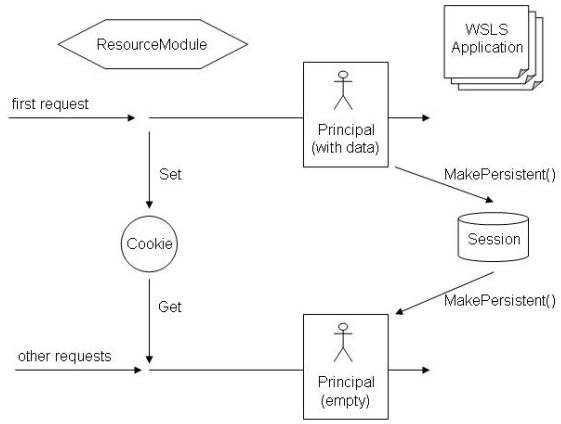

Figure 11: Interface to the resource

7 Logging Events

- 101: user signed-in
- 102: failed authentication
- 103: new user registered
- 104: email sent
- 105: password set back
- 106: user details modified
- 107: email address modified
- 108: password changed
- 109: cronjob executed
- 110: registration re-submitted