www.egiz.gv.at
E-Mail: post@egiz.gv.at
Telefon: ++43 (316) 873 5514
Fax: ++43 (316) 873 5520
Inffeldgasse 16a / 8010 Graz / Austria

MOCCA
Modular Open Citizen Card Architecture

Overview for Developers

Clemens Orthacker *Martin Centner* *Wolfgang Bauer*

Version 1.0 2008-08-29

Table of Contents

1 Glossary

CCE	Citizen Card Environment
CC	Citizen Card
OA	Online Application: a CC enabled Web-Application

2 Introduction

The Austrian citizen card forms the basis for e-government applications. The CC is used for identity management and for qualified electronic signatures. To interact with this CC a middleware, the so called citizen card environment, provides a high-level abstraction layer. Applications use this middleware to make use of the CC's functions. In principle, CCE runs as a service, listening on a dedicated TCP port. This document assumes that you have already studied the specifications of the CCE that can be found on the web (1).

Current CCE implementations have to be installed on the user's PC. This has the significant drawback that the user must install a piece of software before she can use a CC enabled application. In particular this can be annoying in situations, where the user has not the privileges to install new software (e.g. in Internet Cafes or airport lounges).
To overcome this situation, the goal of this project is to develop 3 different forms of the CCEs, which will be described subsequently.

2.1 Local CCE

The local CCE is the conventional approach, where the middleware runs completely on the client's PC. This situation is drawn in the following figure.

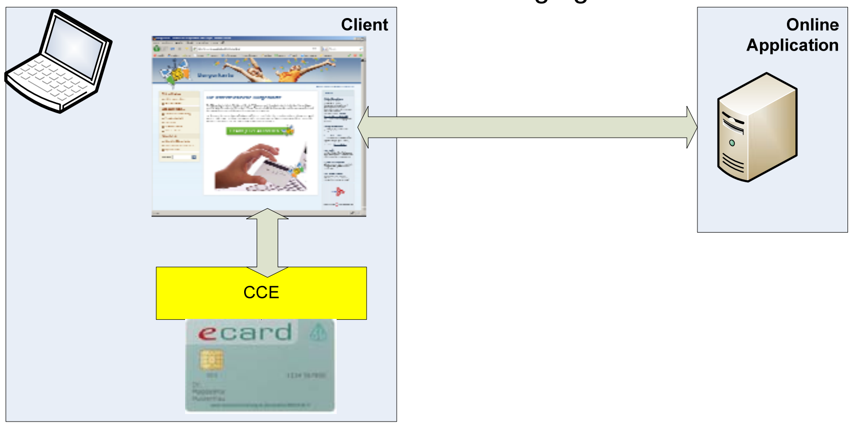


Figure 1: Local CCE

2.2 Online CCE

Whereas the local CCE requires a piece of software to be installed on the client's local PC the online CCE overcomes this shortcoming by a slightly different approach as shown in the following figure.

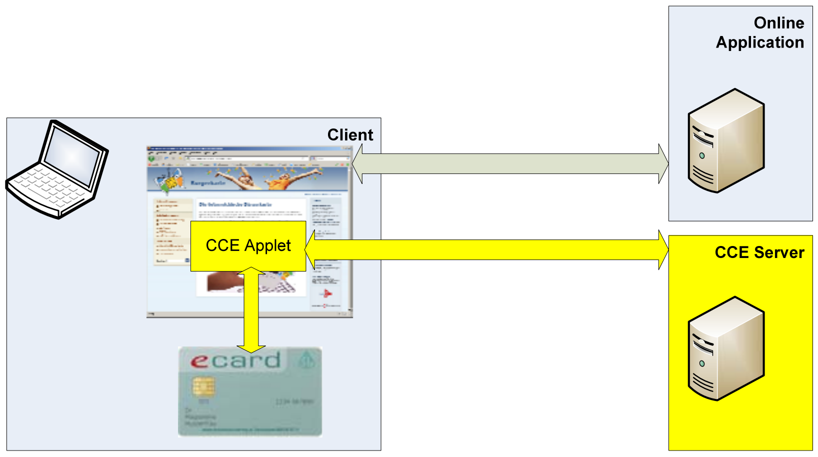


Figure 2: Online CCE

Only the minimum piece of software, required to access the user's CC, runs on the client's machine. To execute this software from the browser it is implemented as Java applet.

2.3 Server CCE

Finally, the server CCE does not require any code to be executed on the client's machine.

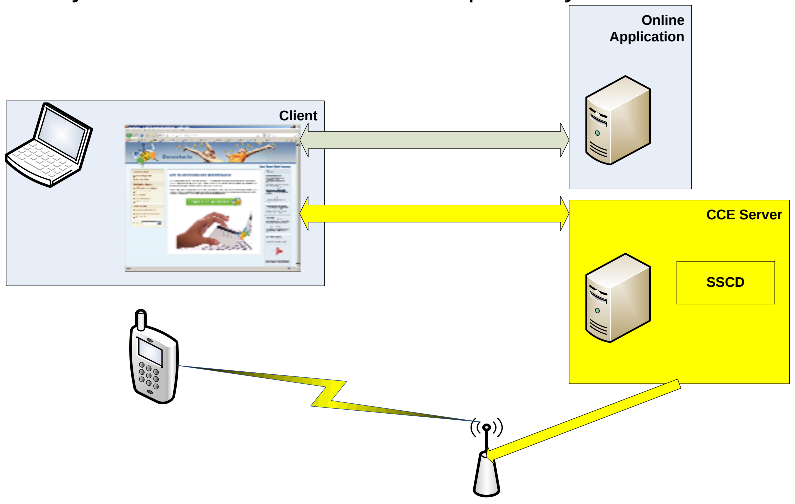


Figure 3: Server CCe

The Server CCE variant does not even require a traditional CC in form of a smart card. However, the users' CC data (private keys, certificates and identity link) are stored on the CCE server.
MOCCA is a software framework that offers support for all three variants. Furthermore, it contains subprojects that use this framework to implement each of the previously described solutions. This document describes the core features and main architectural decisions for the MOCCA project.

3 Requirements

Only the main requirements, significantly influencing the software architecture, are listed here.

3.1 Supported CCE Request

Not all requests as specified in (1) are supported by MOCCA. Only the most relevant, especially when used as server or online CCE, are supported in this project, as listed below:

- Create XML Signature Request
- Infobox Read Request : here only the following infoboxes are supported:

- IdentityLink

To support these requests the CCE must also handle the following infoboxes internally:

- CertifiedKeypair
- SecureSignatureKeypair

Furthermore the following restrictions apply:

No	Description	Remarks
1	SignatureEnvironment/@Reference: only formdata, http/https URL are supported. Furthermore the Response must be of content type text/xml	
2	External Stylesheet or DTD URIs: only HTTP/HTTPS URLs are supported	
3	LocRefContent: only formdata HTTP HTTPS URL supported. In particular this means no relative URIs are supported	
4	Transforms: no stylesheets with references to other stylesheets are supported	Hence we do not support supplements within DataobjectInfo elements

Table 1: Request Restrictions

3.2 *Essential Use Cases*

The main uses cases, defining the software architecture are shown in the following use case diagram.

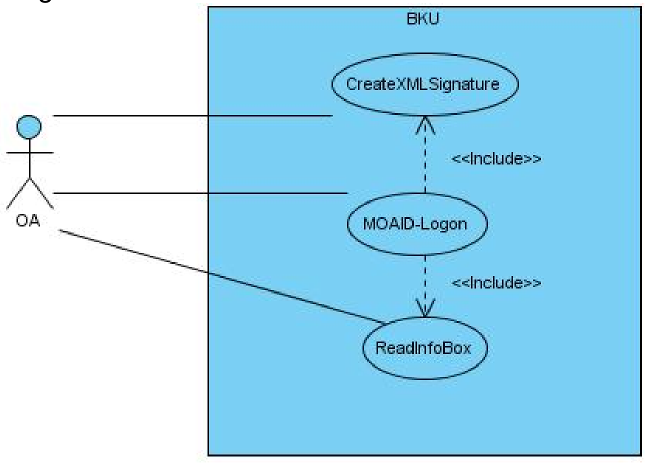


Figure 4: Use Case Diagram

The basic request processing is defined by the CCE's HTTP binding as specified in (2). This specification is translated into the following UML activity diagram.

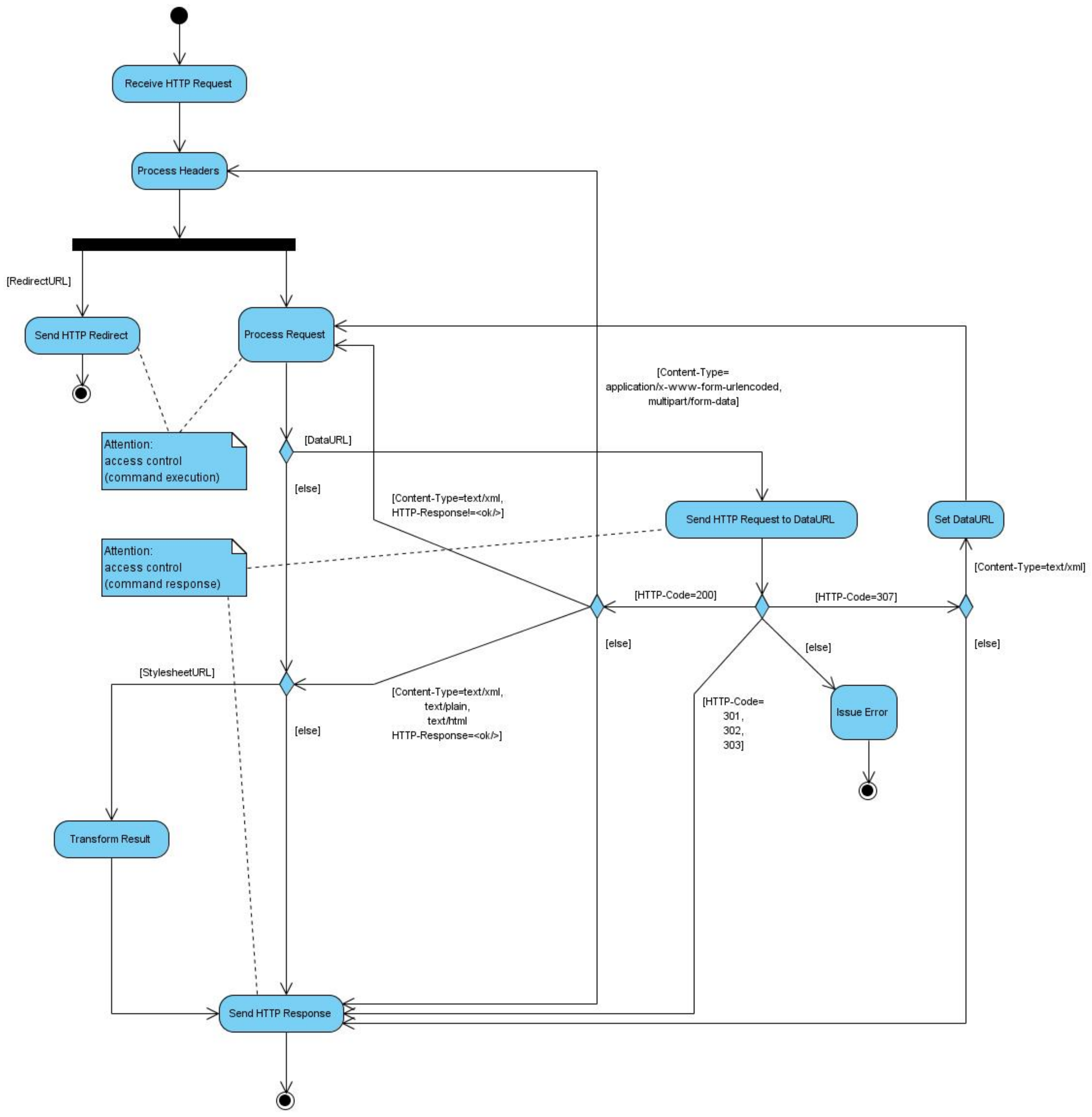


Figure 5: HTTP Binding

4 Software Architecture

4.1 Layering

The MOCCA project uses 3 different abstraction layers as shown in the following figure.

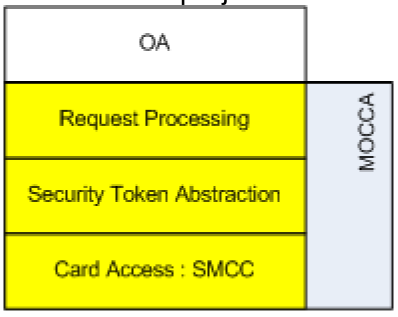


Figure 6: Software Layering

The yellow drawn parts of Figure 6 are implemented in MOCCA. The online application OA accesses MOCCA via the standardized CCE interface. The MOCCA layers have the following responsibilities:

- Request Processing: This layer handles the processing of the XML requests and covers the HTTP binding as described in Figure 5. This layer is intended to be used for all 3 variants of the CCE.
- Security Token Abstraction: this layer is used during request processing to access the CC. Here different implementations exist for local, online and server CCEs.
- SMCC: This smart card abstraction layer is used to communicate with the CC.

4.2 Components

The following component diagram shows the online CCE, which is the most complex scenario. The Browser communicates with the OA. If required the browser invokes the CCE using the standardized CCE interface. The BindingProcessor performs the XML processing and handling of the HTTP binding (e.g. dataurl connection handling). The BindingProcessor uses the STAL to access the security token. In the case of the online CCE, STAL requests are forwarded to the applet (component labeled STAL RequestBroker) using standard web service calls. The applet uses the SMCCStal to access the smartcard on the client's machine (remark: in the local CCE this SMCCStal is directly invoked by the BindingProcessor and thus also a shared component).

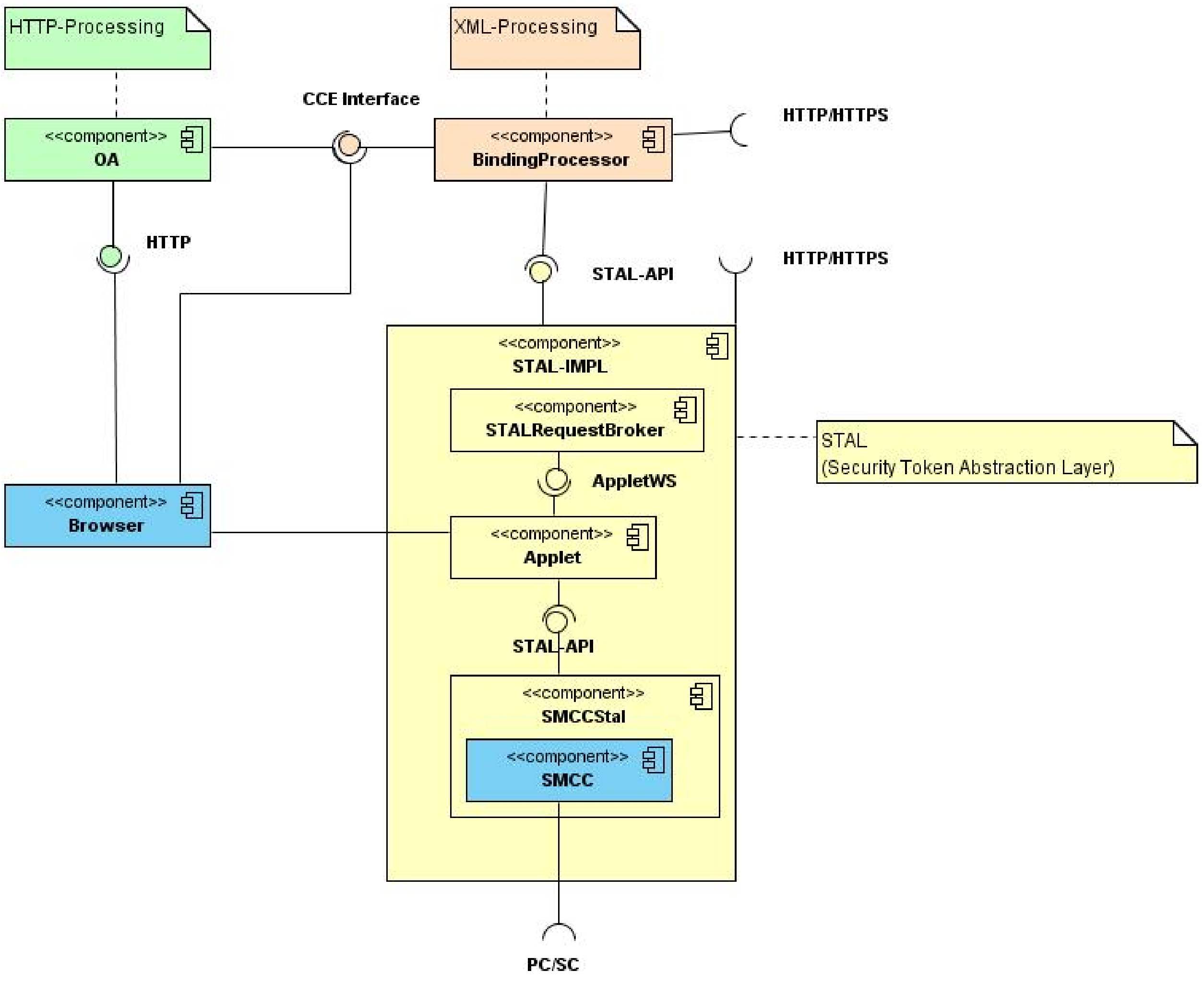


Figure 7: Component Diagram

4.3 Basic Request Processing

The basic request processing is sketched in the following sequence diagram.

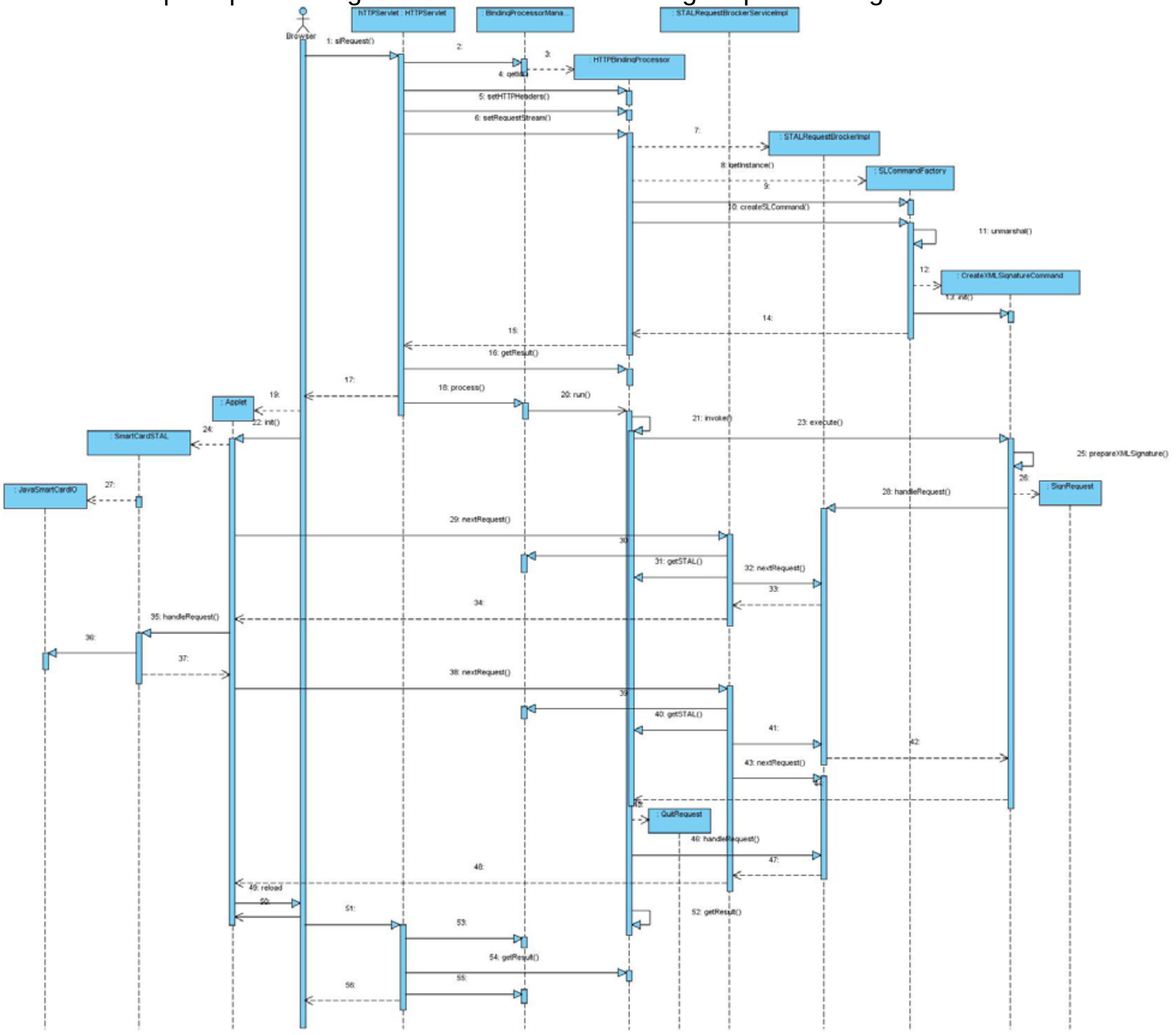


Figure 8: Request Processing

4.4 Packaging

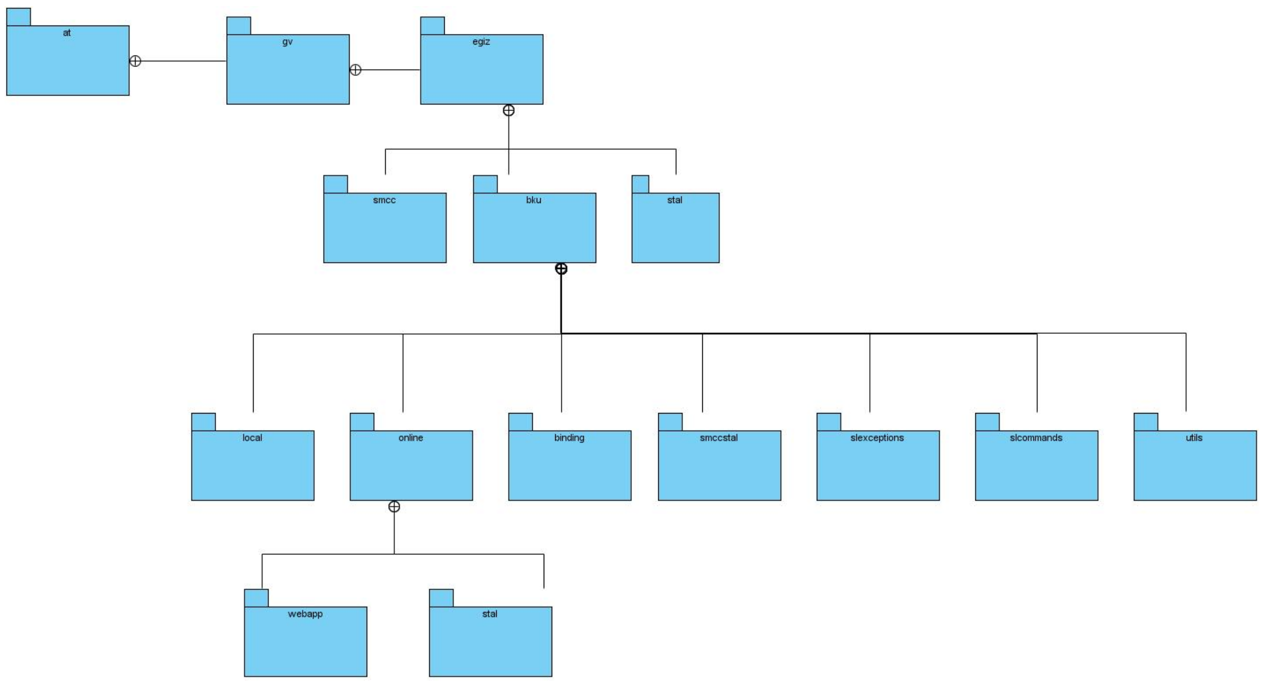


Figure 9: MOCCA Packaging

4.5 Project Structure

The following diagram shows the project structure and dependencies between sub-projects. Each shown “package” is a maven project/module.

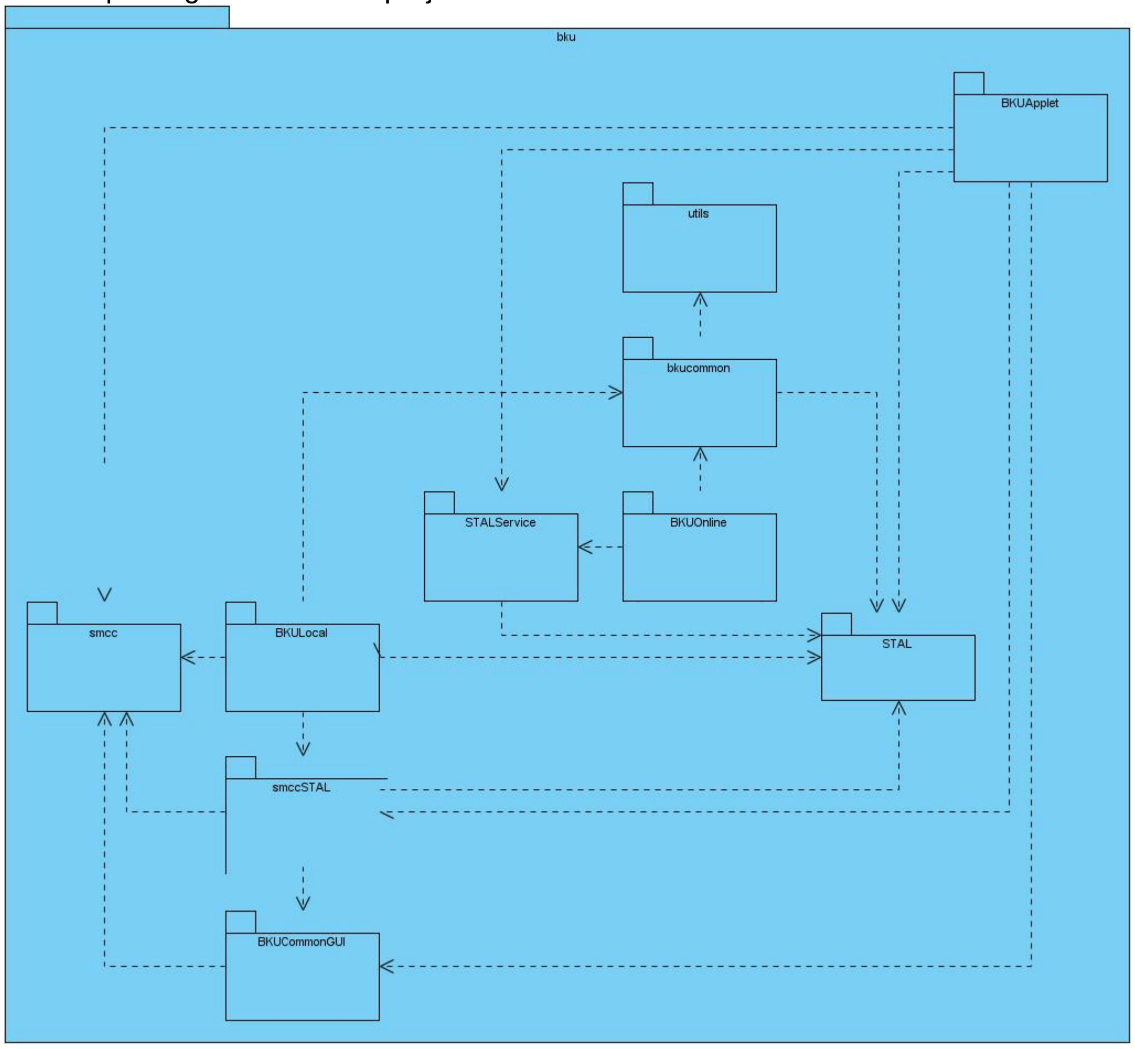


Figure 10: Project Structure

4.6 *Class Diagrams*

The following diagrams show some implementation aspects as class diagram. They are not further commented but should help the reader to study the sourcecode.

4.6.1 HTTP Binding

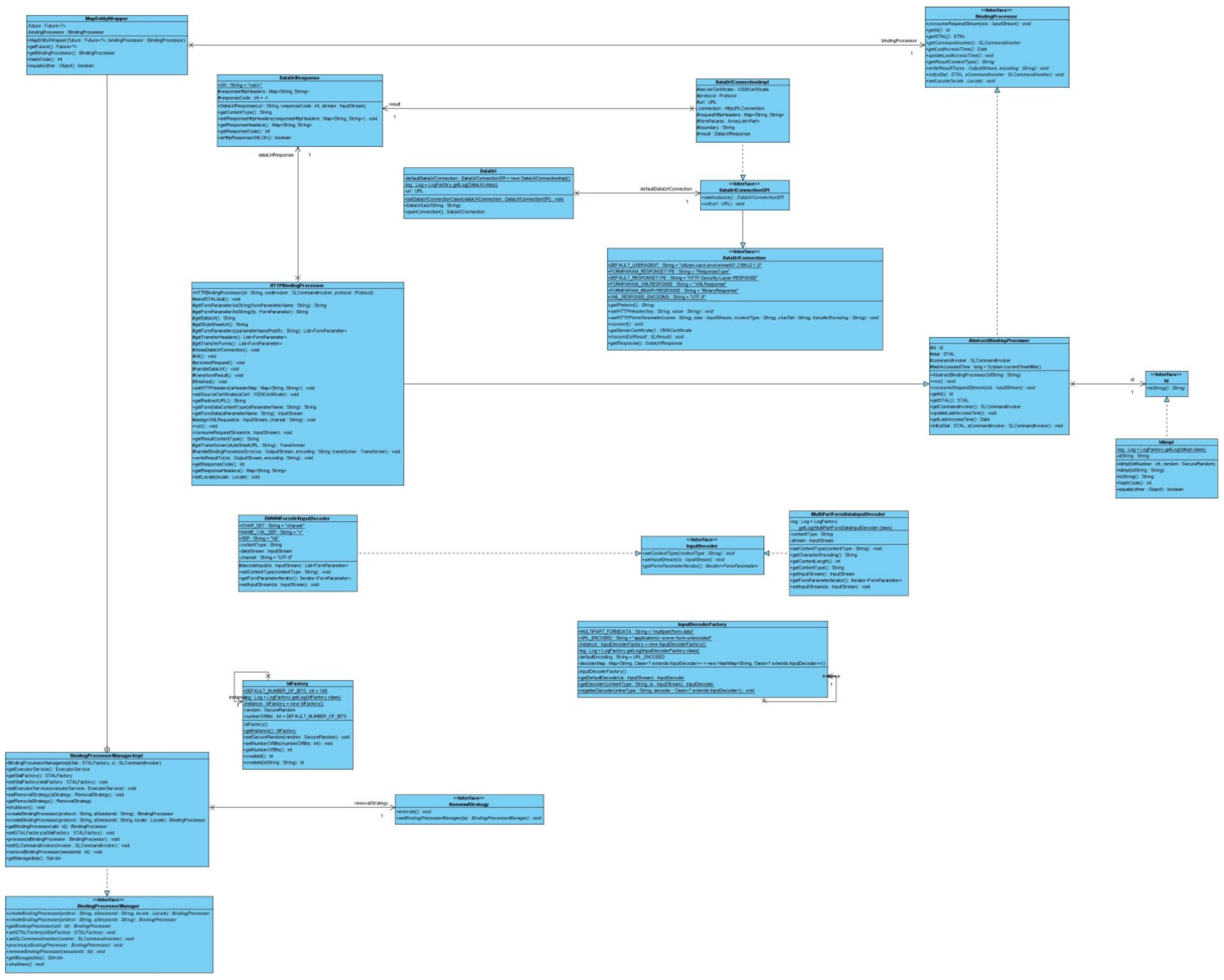
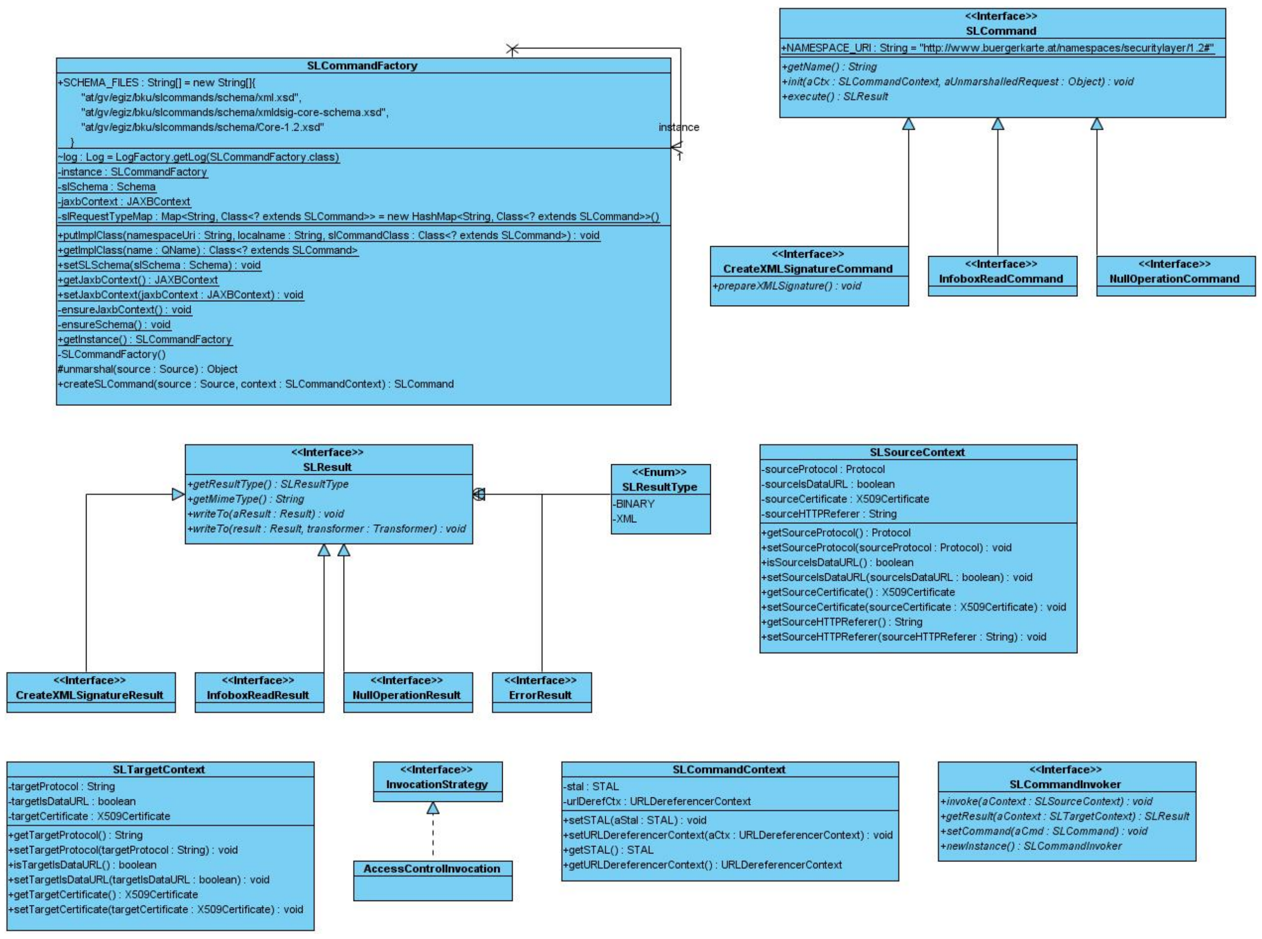


Figure 11: HTTP Binding

4.6.2 SLCommands


SLCommandFactory
+SCHEMA_FILES : String[] = new String[]{
"at/gv/egiz/bku/slcommands/schema/xml.xsd",
"at/gv/egiz/bku/slcommands/schema/xmldsig-core-schema.xsd",
"at/gv/egiz/bku/slcommands/schema/Core-1.2.xsd"
}
~log : Log = LogFactory.getLog(SLCommandFactory.class)
-instance : SLCommandFactory
-slSchema : Schema
-jaxbContext : JAXBContext
-slRequestTypeMap : Map<String, Class<? extends SLCommand>> = new HashMap<String, Class<? extends SLCommand>>()
+putImplClass(namespaceUri : String, localname : String, slCommandClass : Class<? extends SLCommand>) : void
+getImplClass(name : QName) : Class<? extends SLCommand>
+setSLSchema(slSchema : Schema) : void
+getJaxbContext() : JAXBContext
+setJaxbContext(jaxbContext : JAXBContext) : void
-ensureJaxbContext() : void
-ensureSchema() : void
+getInstance() : SLCommandFactory
-SLCommandFactory()
#unmarshal(source : Source) : Object
+createSLCommand(source : Source, context : SLCommandContext) : SLCommand
instance
1
<<Interface>>
SLCommand
+NAMESPACE_URI : String = "http://www.buergerkarte.at/namespaces/securitylayer/1.2#"
+getName() : String
+init(aCtx : SLCommandContext, aUnmarshalledRequest : Object) : void
+execute() : SLResult
<<Interface>>
CreateXMLSignatureCommand
+prepareXMLSignature() : void
<<Interface>>
InfoboxReadCommand
<<Interface>>
NullOperationCommand
<<Interface>>
SLResult
+getResultType() : SLResultType
+getMimeType() : String
+writeTo(aResult : Result) : void
+writeTo(result : Result, transformer : Transformer) : void
<<Enum>>
SLResultType
-BINARY
-XML
SLSourceContext
-sourceProtocol : Protocol
-sourceIsDataURL : boolean
-sourceCertificate : X509Certificate
-sourceHTTPReferer : String
+getSourceProtocol() : Protocol
+setSourceProtocol(sourceProtocol : Protocol) : void
+isSourceIsDataURL() : boolean
+setSourceIsDataURL(sourceIsDataURL : boolean) : void
+getSourceCertificate() : X509Certificate
+setSourceCertificate(sourceCertificate : X509Certificate) : void
+getSourceHTTPReferer() : String
+setSourceHTTPReferer(sourceHTTPReferer : String) : void
<<Interface>>
CreateXMLSignatureResult
<<Interface>>
InfoboxReadResult
<<Interface>>
NullOperationResult
<<Interface>>
ErrorResult
SLTargetContext
-targetProtocol : String
-targetIsDataURL : boolean
-targetCertificate : X509Certificate
+getTargetProtocol() : String
+setTargetProtocol(targetProtocol : String) : void
+isTargetIsDataURL() : boolean
+setTargetIsDataURL(targetIsDataURL : boolean) : void
+getTargetCertificate() : X509Certificate
+setTargetCertificate(targetCertificate : X509Certificate) : void
<<Interface>>
InvocationStrategy
AccessControlInvocation
SLCommandContext
-stal : STAL
-urlDerefCtx : URLDereferencerContext
+setSTAL(aStal : STAL) : void
+setURLDereferencerContext(aCtx : URLDereferencerContext) : void
+getSTAL() : STAL
+getURLDereferencerContext() : URLDereferencerContext
<<Interface>>
SLCommandInvoker
+invoke(aContext : SLSourceContext) : void
+getResult(aContext : SLTargetContext) : SLResult
+setCommand(aCmd : SLCommand) : void
+newInstance() : SLCommandInvoker

5 Bibliography

1. **Arno Hollosi, Gregor Karlinger, Thomas Rössler, Martin Centner, et al.** *Die Applikationsschnittstelle Security-Layer zur österreichischen Bürgerkarte.* 2008.

2. —. *Transportprotokolle für die Applikationsschnittstelle Security-Layer der österreichischen Bürgerkarte.* 2008.

3. *Bürgerkartenspezifikation.* [Online] 2008. http://www.buergerkarte.at/konzept/securitylayer/spezifikation/aktuell/.