Brokerage in web application frameworks

Issues of framework re-use in OpenACS

Stefan Sobernig
Vienna University of Economics and Business Administration

Overview

1. On concepts: brokerage and framework re-use
2. Framework re-use by XOTcl Request Broker (xorb)
3. Looking ahead ...

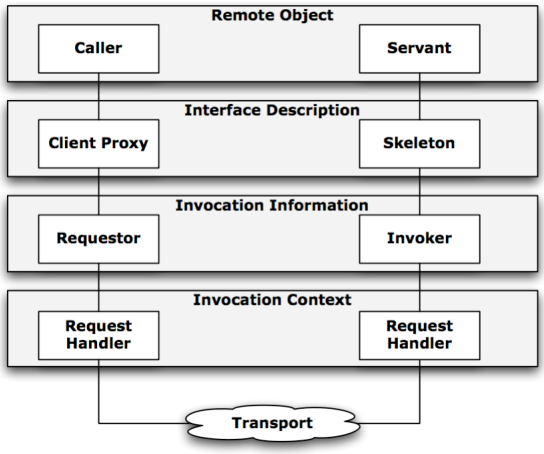
On concepts / Broker

A broker?
Brokering what?

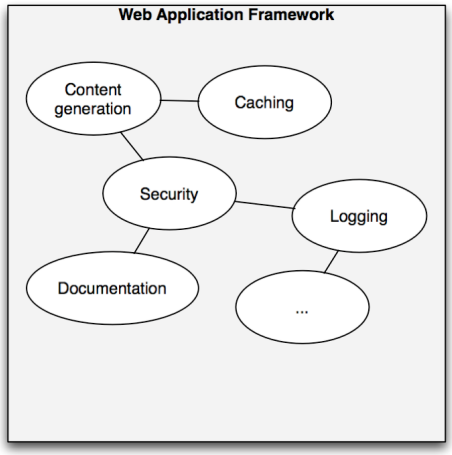
On concepts / Broker as architectural pattern

- The key metaphor in engineering & designing an support infrastructure for distributed applications.
- An explicit architectural form characteristic to the context of distributed applications [3, 2, 1].
- The broker includes a set of characteristic constituents, grouped into layers. They are either members to a layer or inter-layer siblings.

On concepts / Broker as pattern composition

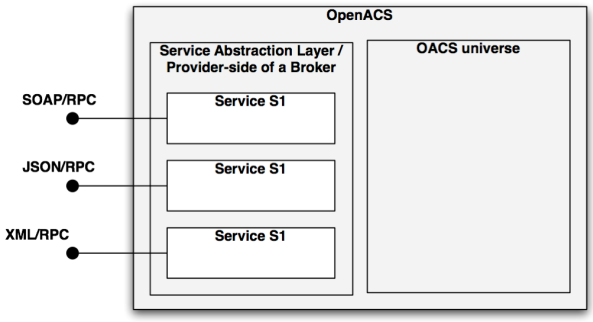


On concepts / Web application framework

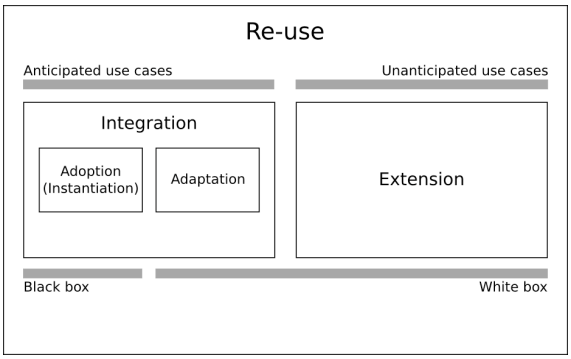


On concepts / A missing link?

- Web applications are, sui generis, characterised by the expected multiplicity of interaction and delivery channels; the idea of a Service Abstraction Layer (Vogel 2002).



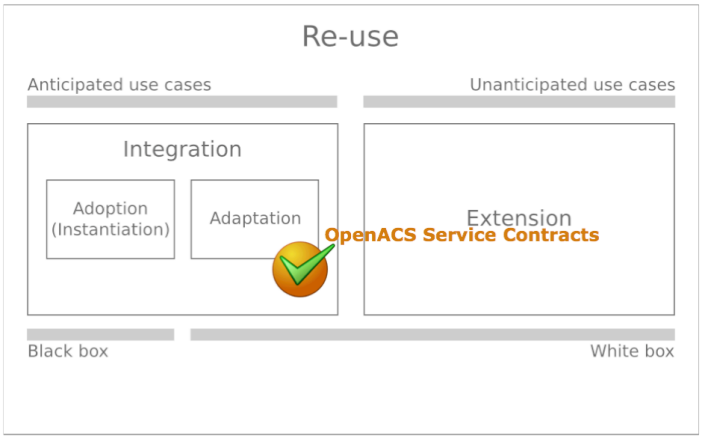
On concepts / Kinds of framework re-use



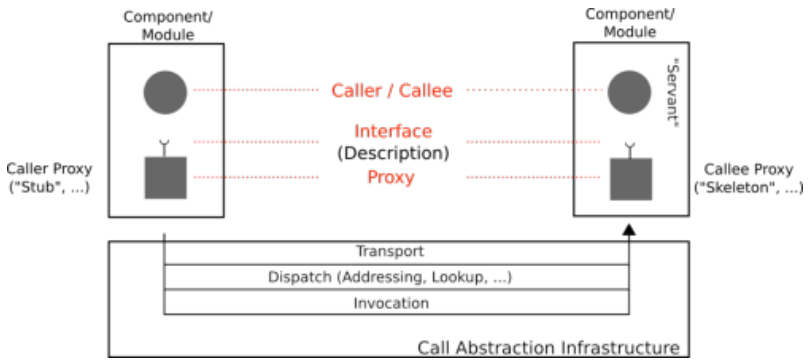
Framework re-use by XOTcl Request Broker

- Generic brokerage (Völter et al. [2005]) infrastructure for OpenACS, based on XOTcl and xotcl-core.
- Allows for plugging-in protocol extensions: currently SOAP support by xosoap.
- Based upon an object-oriented layer and extension to OpenACS service contracts, allowing for a more agile use of contracts and implementations.
- Allows for publishing existing Tcl and XOTcl code as remoting, e.g. SOAP, services.
- Support for legacy code through ”interface adapters”
- Generic extension mechanism through ”invocation interceptors”
- Fine-grain facilities for invocation access control
- Tight integration with XOTcl idioms

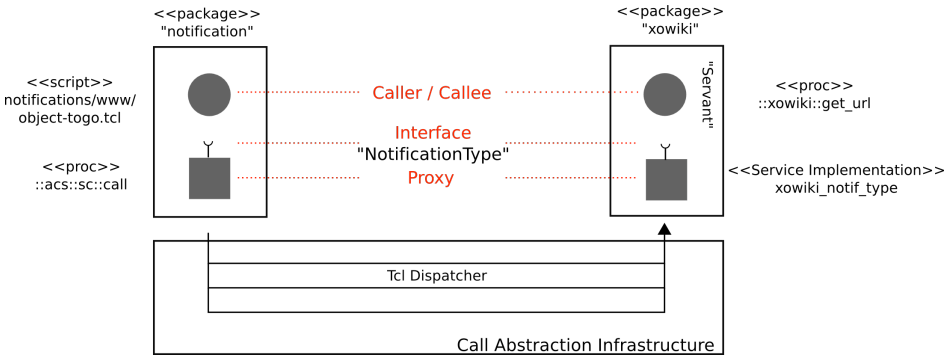
Re-use / OpenACS Service Contracts (1)



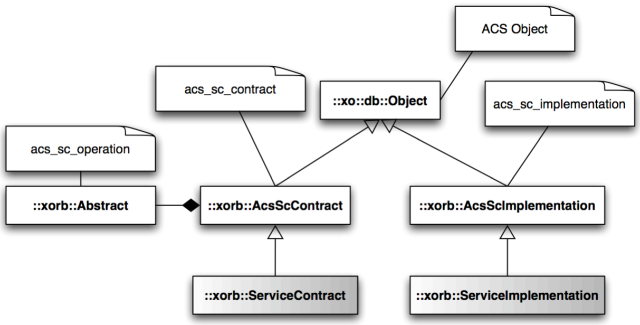
Re-use / OpenACS Service Contracts (2a)



Re-use / OpenACS Service Contracts (2b)



Re-use / OpenACS Service Contracts (3)

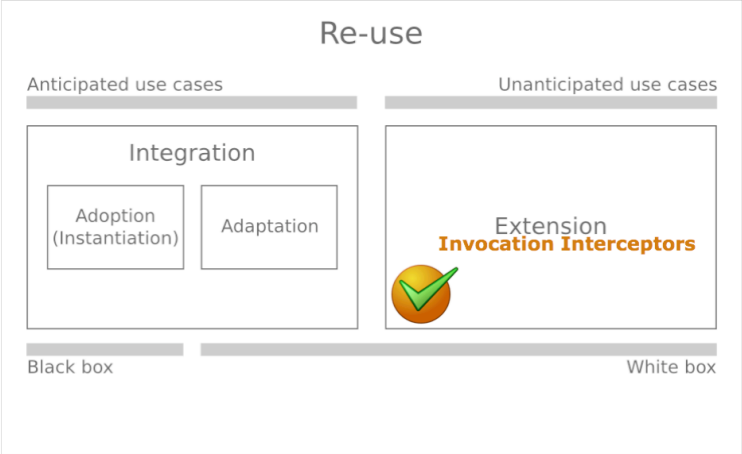


Re-use / OpenACS Service Contracts (4)

The XOTcl Request Broker settles between adaptation and extension

- Adaptation:
  - Adopts the persistence strategy for OpenACS Service Contracts (interface descriptions) and Implementations (skeletons).
  - Integrates with the administration and maintenance interfaces (/acs-service-contract).
  - More agile usage of contract and implementations of objects.
- Extension:
  - A refined invocation dispatcher, i.e. the Invoker.
  - Materialises the Service Abstraction Layer idea.

Re-use / Invocation Interceptors

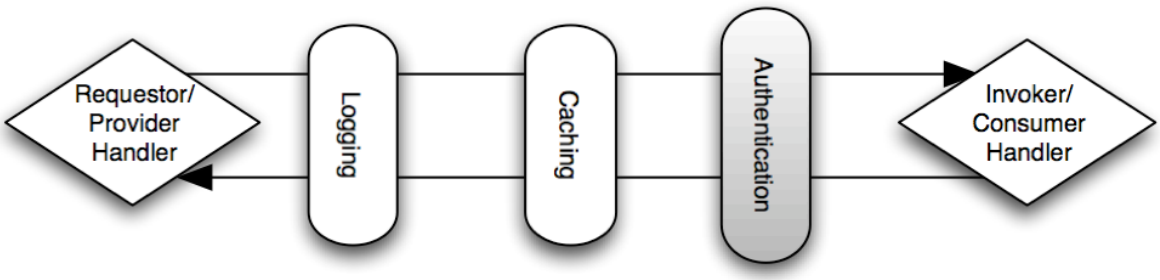


Re-use / Invocation Interceptors (2)

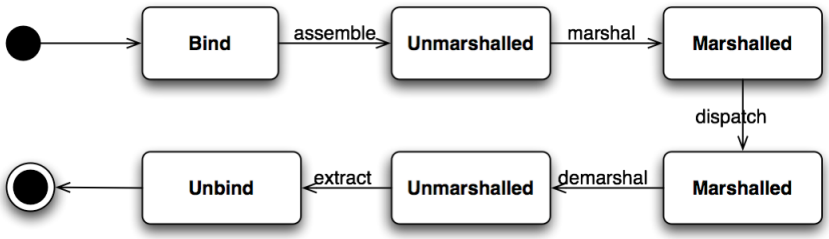
... are means to extend the overall functionality by ser vices by tasks that are considered orthogonal or rather dynamic.

- Interceptors are hooks along the path the arriving invocation data passes through the request broker. A chain of interceptors is the required infrastructure to provide to enforce these hooks.
- Interceptors realise functionality that is considered add-on, for example various security-related tasks: authentication/ identity, confidentiality and integrity verification.
- They listen along the invocation path and can access the overall invocation data, modify it, etc.
- They are realised using XOTcl mixin chains
- Interceptors can be configured to listen to various scopes: (1) protocols, (2) implementations.

Re-use / Invocation Interceptors (3)



Re-use / Invocation Interceptors (4)

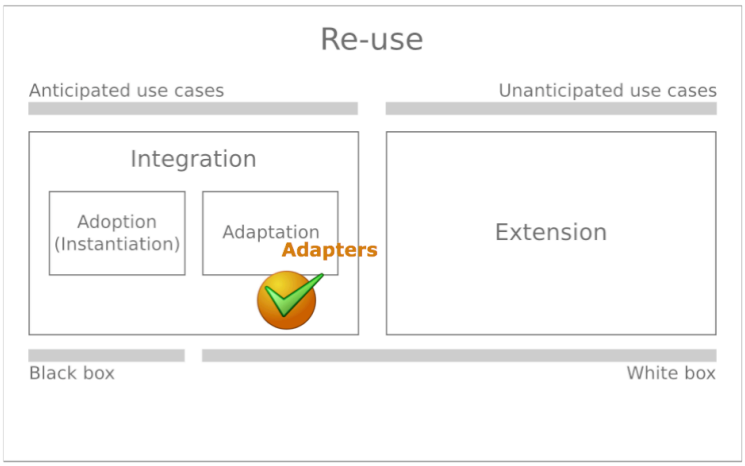


Re-use / Invocation Interceptors (5)

Skeleton of an authentication interceptor:

```
SoapInterceptor AuthenticationInterceptor

AuthenticationInterceptor proc checkPointcuts {context} {
    # apply?: 0 or 1
}

AuthenticationInterceptor instproc handleRequest {context} {
    # proceed with authentication
}

AuthenticationInterceptor instproc handleResponse {context} {
    # clear authentication state
}

# -- register
::xorb::provider_chain add [AuthenticationInterceptor self]
```


Re-use / Adapters



Re-use / Adapters (2)

... are means to expose existing code as service implementations without changing
the original interfaces

- 3 types: Object-, Class- and ProcAdapter.
- They are 2-in-1 special-purpose objects: They realise a service implementation in the background while providing interface bridges to legacy servant code.
- This role is best captured by the Adapter pattern description.
- The realisation is based upon techniques ranging from simple inheritance to advanced XOTcl interception techniques, i.e. filters.

Re-use / Adapters (3)

```
# / / / / / / / / / / / / / / / /
# 1) declare adapter class

  ProcAdapter SearchAdapter \
      -implements SearchService \
      -adapts {
        synchronousQuery   ::tsearch2::search
      }
# / / / / / / / / / / / / / / / /
# 2) provide for a signature adapter
# between synchronousQuery and
# ::tsearch2::search

  SearchAdapter instproc search {
    -targetSessionID:string
    -queryStatement:string
    -startResult:integer
  } {
    next;# -> ::tsearch2::search
  }
```

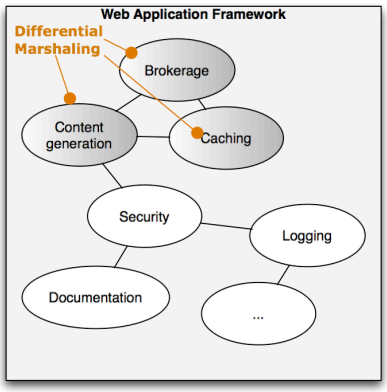
Looking ahead ...

Un-anticipated / Differential Marshaling (1)

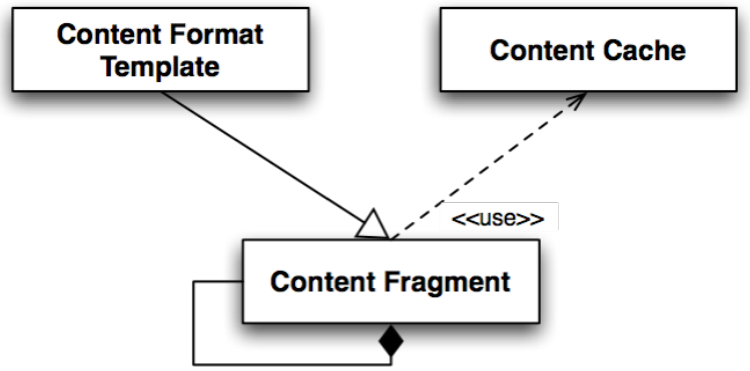
Differential marshaling refers to ...

- Distinguishing between mutable/immutable elements of transport messages (SOAP/XML)
- Apply content generation optimisation techniques such as caching and templates to re-use immutable parts.
- This issue has been prominently discussed in Web Service related literature to tackle the severe overhead and scalability issues related XML processing.
- There are many empirical findings available, they all report considerable improvements compared to redundant marshaling, e.g. by a factor of 10+ (processing time) compared to ordinary DOM-2 marshaling, for instance.

Un-anticipated / Differential Marshaling (2)



Un-anticipated / Differential Marshaling (3)

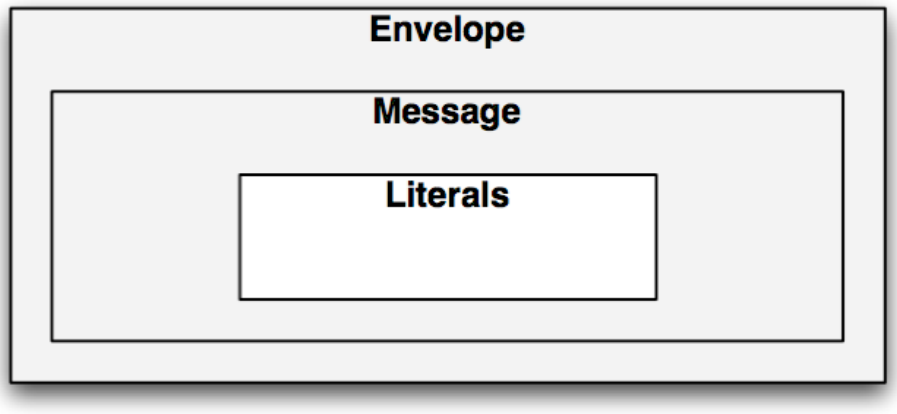


Un-anticipated / Differential Marshaling (4)

A sample SOAP/XML message document:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<ns0:getSelectedQueryResponse xmlns:ns0="http://bpms.intalio.com/tools/webservi
    <ns0:getSelectedQueryResult>
        <ns0:Order>
            <ns0:Id>123</ns0:Id>
            <ns0:Price>EUR123</ns0:Price>
        </ns0:Order>
    </ns0:getSelectedQueryResult>
</ns0:getSelectedQueryResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```


Un-anticipated / Differential Marshaling (5)



Un-anticipated / Differential Marshaling (6)

An approach in the sense of Content Format Templates is to ...

- Generate a template for the Envelope and Message parts
- Using a templating engine, i.e. OpenACS' ADP
- Provide for a mapping of concepts between XML Document Models to template expressions
- Finally, compile the template once (upon declaration time) and re-use it for all marshaling tasks on this service.
- This realises differential marshaling by means of re-using content generation and delivery techniques provided by a web application framework.

Un-anticipated / Differential Marshaling (7)

Turning the Document into an ADP Template:

```
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://schemas.xmlsoap.org/soap/envelope/">
<SOAP-ENV:Body>
<ns0:getSelectedQueryResponse xmlns:ns0="http://bpms.intalio.com/tools/webservi
    <ns0:getSelectedQueryResult>
        <ns0:Order>
            <ns0:Id>@id@</ns0:Id>
            <ns0:Price>@price@</ns0:Price>
        </ns0:Order>
    </ns0:getSelectedQueryResult>
</ns0:getSelectedQueryResponse>
</SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Un-anticipated / Differential Marshaling (8)

The way of proceeding:

- We assume that you have an in-memory rep of the message, i.e. an XOTcl object
- The object's class defines the overall message structure (by taking into account XML specifics)
- Upon declaration of the class, you may opt to generate an ADP template from class abstract state (properties/slots)
- Compile the ADP template (adp_compile)
- Upon request handling, the message object is handed over to the Marshaller to be streamed in its XML representation.
- In this process, the Marshaller looks-up the template registered with the class, and ...
  - Gather literal information from a Data Update Table

  - Evaluate the template against the instance data: adp_eval
- Voilà ...

Bibliography

- Markus Völter, Michael Kircher, and Uwe Zdun. Remoting Patterns: Foundations of Enterprise, Internet and Realtime Distributed Object Middleware. Software Design Patterns. John Wiley & Sons Ltd., Chichester, England, 2005
- Michael Kircher, Markus Völter, Klaus Jank, Christa Schwanninger, and Michael Stal. Broker revisited. In Proceedings of EuroPLoP 2004, Irsee, Germany, 2004
- Frank Buschmann, Regine Meunier, Hans Rohner t, Peter Sommerlad, and Michael Stal. Pattern-Oriented Software Architecture – A System of Patterns, chapter Broker, pages 99–122. John Wiley & Sons Ltd., Chichester, England, 2000