Design and Implementation of an Asynchronous Invocation Framework for Web Services

Uwe Zdun
zdun@acm.org

Markus Voelter
voelter@acm.org

Michael Kircher
michael.kircher@siemens.com

The International Conference on Web Services - Europe 2003 (ICWS-Europe'03), Erfurt, Germany, Sep 2003.

Overview

- Synchronous vs. asynchronous communication
- Asynchronous invocation of Web Services
- Client Asynchrony Patterns
- Simple Asychronous Invocation Framework for Web Services

Synchronous vs. Asynchronous Communication

- Synchronous communication in remote object frameworks:
  - The client wants to reach a Remote Object
  - It invokes a Client Proxy in the client process that handles network communication
  - The client blocks until the Client Proxy returns the result from the Remote Object invocation
- Asychronous communication in remote object frameworks:
  - The client also invokes a Client Proxy, but . . .
  - The Client Proxy returns to the client immediately and handles the remote invocation on its own
  - Different variants how to pass the result (and exceptions) back to the client (see asynchrony patterns).

Asynchronous Invocation of Web Services

- Asynchronous invocations are an important functionality in the context of distributed object frameworks:
    - jitter and network latency make remote invocation times unpredictable
    - in many situations clients should not block during remote invocations
    - loose coupling between clients and remote services
- Popular web service implementations (such as Apache Axis) offer only synchronous invocations (over HTTP) or messaging protocols
- Client asynchrony can be built on top of synchronous invocation framework $\longrightarrow$ Asynchrony Patterns
- This is tedious and error-prone $\longrightarrow$ *Simple Asychronous Invocation Framework for Web Services*

Client Asynchrony Patterns

- A pattern describes a recurring *solution* to a *problem* in a *context* balancing a set of *forces*:
  - Patterns cover the problem that expertise is hard to convey
  - Pattern Languages: no pattern is used in isolation $\longrightarrow$ patterns are used as elements of a language
- Four patterns for client asynchrony from a larger pattern language for OO Remoting
- Full pattern language in forthcoming book “**Remoting Patterns**” by Markus Voelter, Michael Kircher, Uwe Zdun, and Michael Englbrecht to be published in Wiley’s Pattern Series in 2004.

Client Asynchrony Patterns: Fire and Forget/Sync with Server

- Fire and Forget:
  - A Remote Object should be notified and a result is not required
  - Reliability is not critical
  - Client Proxy sends invocation and returns to the client immediately
  - It does not wait for a notification
- Sync with Server:
  - A Remote Object should be notified and a result is not required
  - The invocation should be performed reliably
  - Client Proxy sends invocation and returns to the client immediately
  - It waits for an acknowledgment

Client Asynchrony Patterns: Poll Object/Result Callback

- Poll Object:
  - An operation should be invoked asynchronously and a result is required
  - The client is able to decide when to use the returned result
  - Poll Objects receive the result of remote invocations on behalf of the client
  - The client subsequently uses the Poll Object to query the result
- Result Callback
  - An operation should be invoked asynchronously and a result is required
  - The client needs to react immediately on incoming results
  - The client passes a Result Callback object to the Client Proxy
  - For arriving results the Client Proxy calls the predefined callback operation

Alternatives for Applying the Patterns

Client asynchrony pattern	*Result to client*	*Acknowledgment to client*	*Responsiblity for result*
Fire and Forget	no	no	-
Sync with Server	no	yes	-
Poll Object	yes	yes	Client is responsible for getting the result
Result Callback	yes	yes	Client is informed via a callback

Simple Asynchronous Invocation Framework for Web Services

- Framework that realizes the asynchrony patterns on top of synchronous invocations
- Works with Apache Axis on top of HTTP
- Can be downloaded from: `saiws.sourceforge.net`

Client Proxies

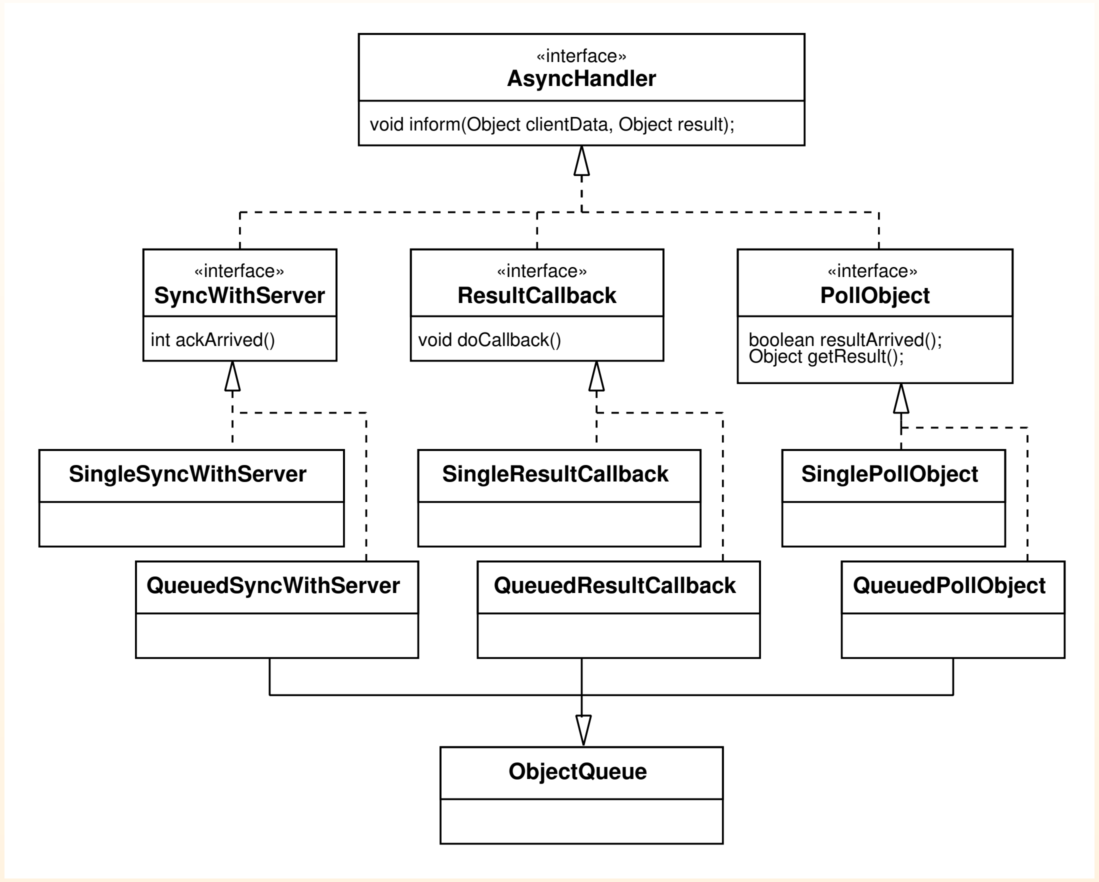
Invocation is performed using a Client Proxy. Synchronous invocations:

```
SyncClientProxy scp = new SyncClientProxy();
String result = (String) scp.invoke(endpointURL,
                  operationName, null, rt);
```

Asynchronous invocation:

```
AsyncHandler ah = ...;
Object clientACT = ...;
AsyncClientProxy ascp = new asyncClientProxy();
...
ascp.invoke(ah, clientACT, endpointURL, operationName,
            null, rt);
```

Asynchrony Handlers

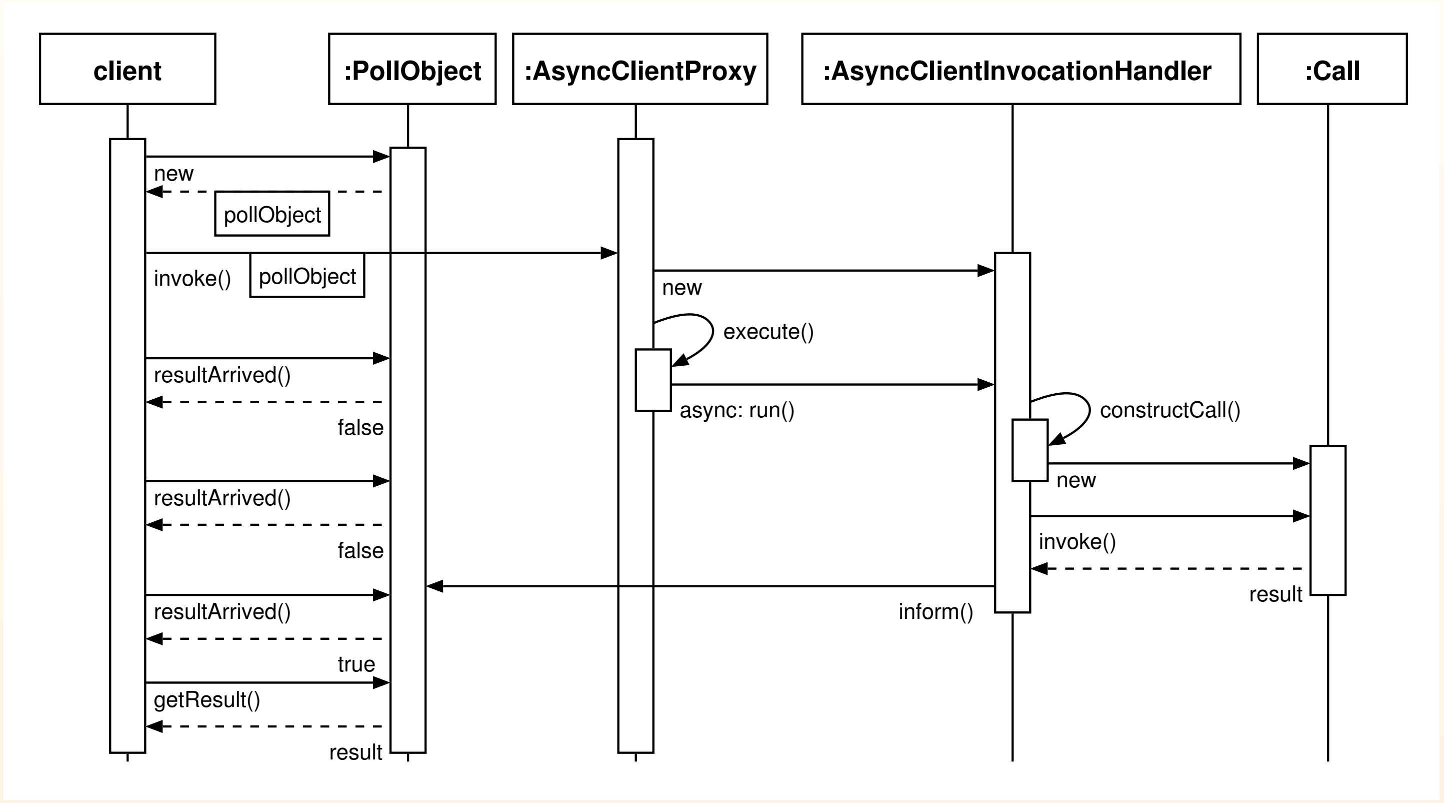


Example: Poll Object

```
AsyncClientProxy clientProxy = new AsyncClientProxy();
SimplePollObject p = new SimplePollObject();
clientProxy.invoke(p, null, endpointURL, operationName,
                   null, rt);

while (!p.resultArrived()) {
  // do some other task ...
}
System.out.println("Poll Object Result Arrived = " +
                   p.getResult());
```

Poll Object Dynamics



Queued Asynchrony Handlers

- Handle multiple responses
- Queuing handlers with FIFO behavior are pre-defined
- Client ACT (Asynchronous Completion Token) identifies invocation

Example: Queued Result Callback

```
AsyncClientProxy clientProxy = new AsyncClientProxy();
DateClientQueue results = new DateClientQueue(10);
for (int i = 0; i < 10; i++) {
  String id = "callback" + i;
  clientProxy.invoke(results, id, endpointURL,
                     operationName, null, rt);
}
```

Fire and Forget Invocations

Fire and Forget is not implemented using an AsyncHandler, but with an operation.

Internally implemented using one-way invocations (as in WSDL).

```
AsyncClientProxy clientProxy = new AsyncClientProxy();
clientProxy.invokeFireAndForget(endpointURL,
                                operationName,
                                null, rt);
```

Performance

Performance Test	Synchronous Invocation	Fire and Forget	Sync with Server	Poll Object	Result Callback
1 invocation	30ms	1ms	1ms	1ms/39ms	1ms/42ms
3 invocation	68ms	2ms	2ms	2ms/89ms	2ms/69ms
10 invocation	204ms	2ms	2ms	2ms/265ms	2ms/189ms
20 invocation	378ms	5ms	4ms	5ms/409ms	4ms/368ms

Conclusion

- Practical approach for asynchronous invocations of web services
    - Simple invocation API
    - Easily extensible with new handlers
- Designed with a set of patterns from a larger pattern language for distributed object frameworks
- The SAIWS framework can be downloaded at: `saiws.sourceforge.net`
- More information on the patterns can be found in the forthcoming book and in our VikingPlop/EuroPlop papers