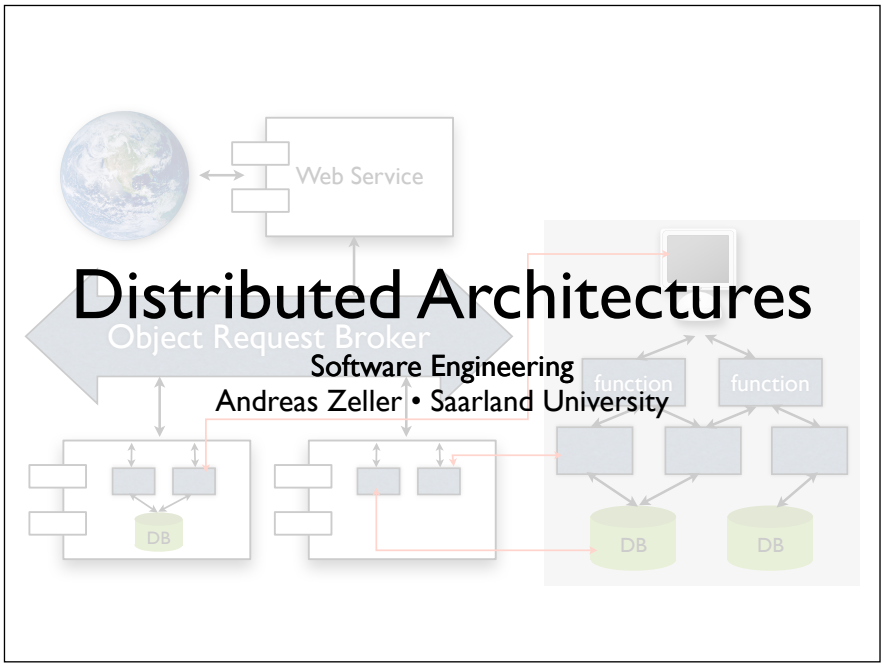
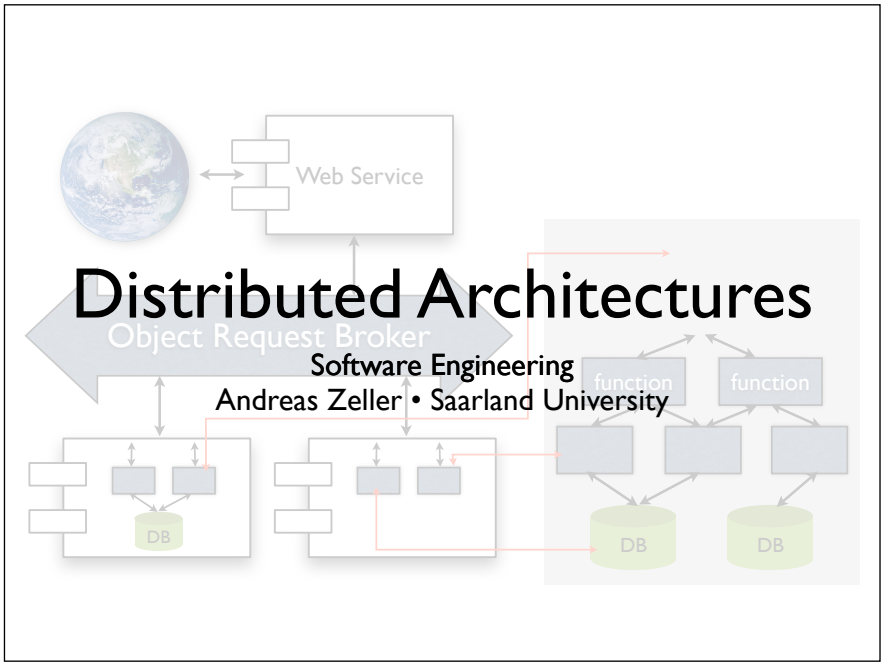


Distributed Architectures

Software Engineering
Andreas Zeller • Saarland University

These slides are based on Wikipedia and other Web sources (URLs given)

1



Distributed Architectures

Software Engineering
Andreas Zeller • Saarland University

These slides are based on Wikipedia and other Web sources (URLs given)

2

Modularity

Modularity, again

3

- but this time at a far greater scale

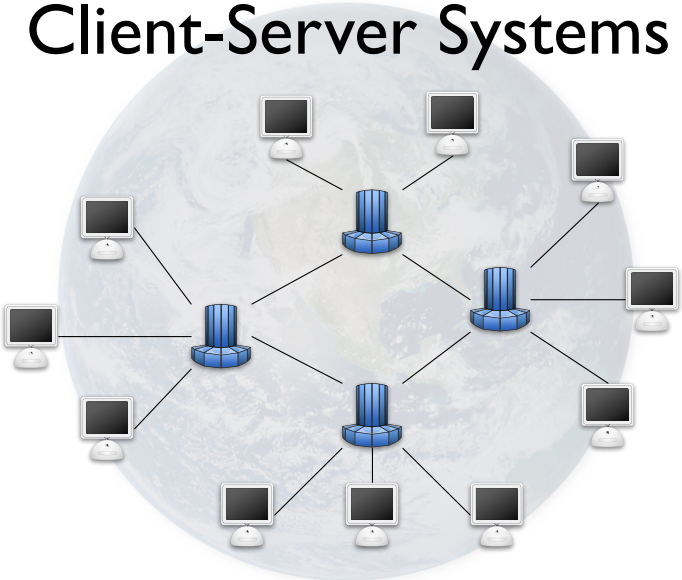
4

The Challenge

- How do we build applications that are *distributed* across the entire globe?

5

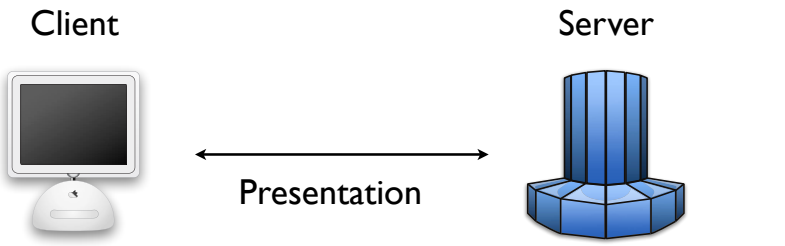
Client-Server Systems



See Sommerville, Software Engineering, 11.2

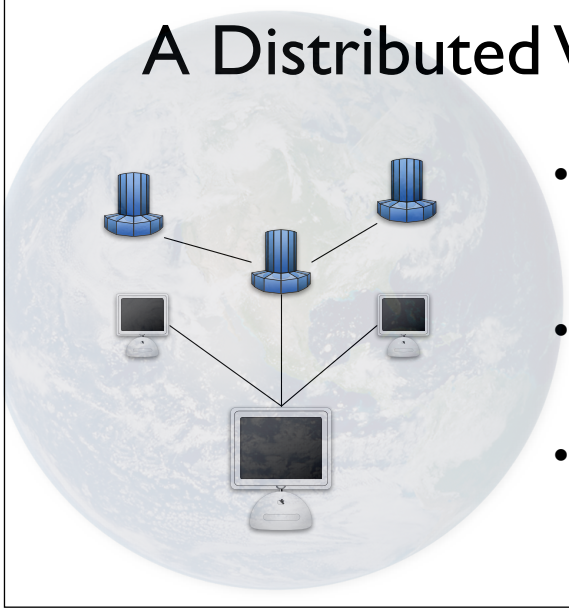
6

Thin Clients



Thin-client model: Everything is processed on the server
Fat-client model: Server only responsible for data management

A Distributed World



- A single client can be connected to several servers (and other clients)
- … and access and combine all of their services
- How do we organize this?

http://en.wikipedia.org/wiki/Distributed_computing

Modules

- *statically linked* to each other at build time
- can no longer be *separated* once composed
- *controlled* by the programmer(s)

Components

- linked, composed, and distributed *dynamically* at run-time
- allow for *arbitrary compositions* of services
- systems can include third-party *components of the shelf* (COTS)

What is a component?

- A component provides predefined *services* (like a module or object)
- A component is
  - *deployed*
  - *composed*
  - *independent*

Deployed Components

A component can be deployed.

- A component can *communicate* with other components
- Communication and deployment can take place over a network

Composed Components

A component can be composed to a system by third parties.

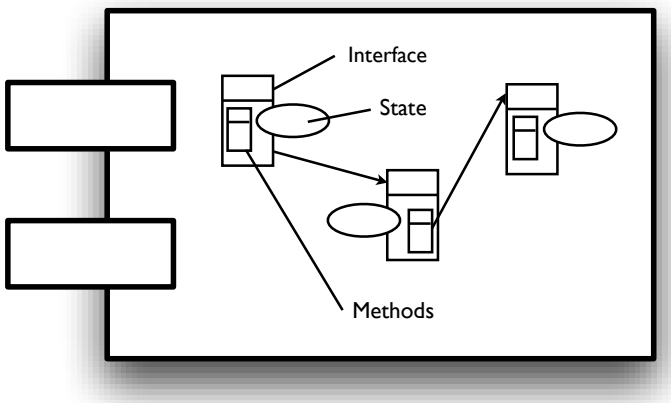
- A component must not require specific *construction steps*
- A component must specify its *requirements* and *services*
- A component must only communicate via its *interface*

Independent Components

A component can be deployed as a unit.

- A component therefore is *separated* from its environment (and other components)
- A component *encapsulates* all it needs
- A component is not deployed in parts

What's in a Component?



A component contains objects with states

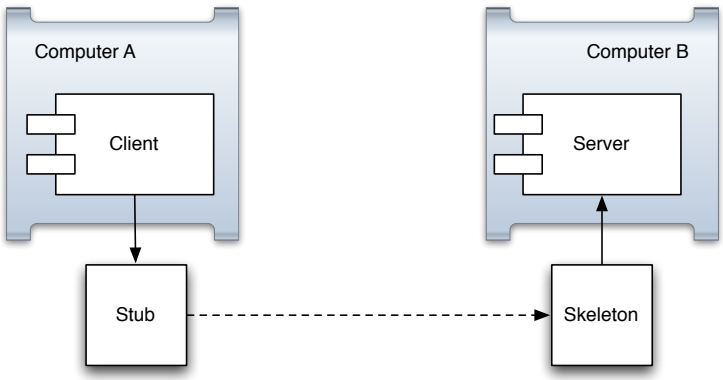
Communication

- Components need *references* to be able to address each other
- Components need *transport mechanisms* to communicate with each other

References

- To communicate with a server, a client needs a *reference* to the server object
- This reference is obtained from another call – or the middleware's *name service*
- References ≠ Pointers
  (as the other component may not be stored in memory)

Communication



- Stub and Skeleton share *common interface*

Stubs and Skeletons

- *Stub* = interface for remote services
  - Can be used just like a local object
- *Skeleton* = code template for services
  - Code template with signatures and placeholders for implementation

Packing and Unpacking

- When the client invokes an operation, the stub *packs* (serializes, marshals) all data and sends it to the computer holding the skeleton object
- The skeleton *unpacks* the data and invokes the actual server operation
- Return values are handled in the same way
- Needs time; but negligible wrt transport

Interface Description

- Stubs and skeletons are *generated* from a common interface description
- Special interface description language (IDL)
- Specification is language-neutral
  and thus allows for communication between components in different languages

Middleware

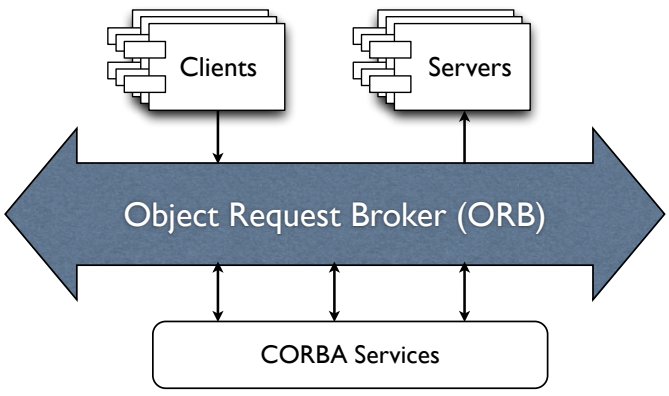
Middleware = software *connecting* components

- The CORBA Standard
- More Standards: Java EE, .NET
- Web Services
- Service-Oriented Architectures

CORBA

- Common Object Request Broker Model
- Oldest Middleware Standard (OMG 1991)
- A specification, not a product

CORBA Architecture

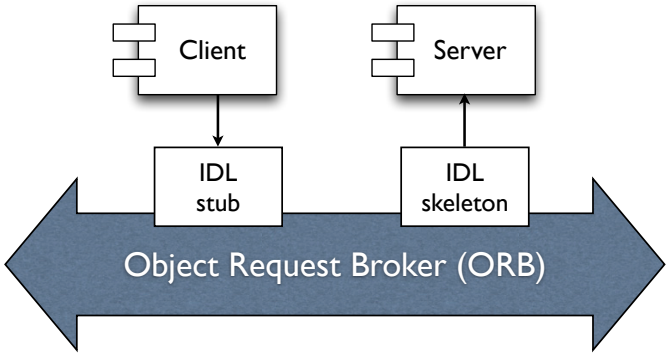


Object Request Broker

The ORB *manages requests for object services:*

- It locates the object providing the service
- It prepares it for the request
- It sends the service request
- It returns the result to the requester

Stubs and Skeletons



The CORBA IDL

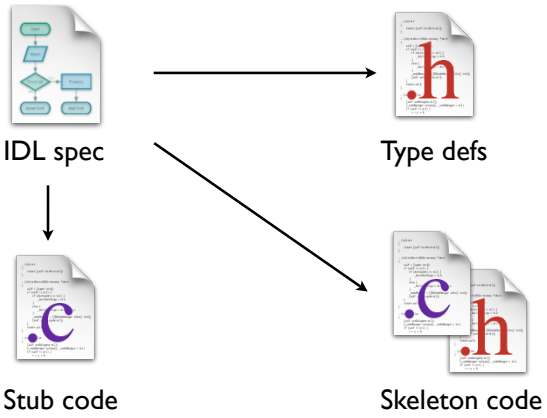
- Describes interface in language-independent manner
- Standardized mappings for all languages (best for C++ and Java)

http://en.wikipedia.org/wiki/CORBA

CORBA IDL Principles

- Objects by Reference
  (Objects by Value is also possible if the code is known or downloadable)
- Data by Value
  (short • long • float • double • char • boolean • string • enum • any…)
- Structures by Value
  (structure • union • array • sequence • exception)

From IDL to Code

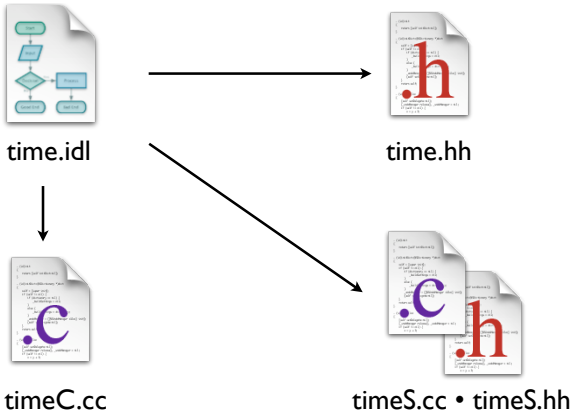


A Time Server Interface


time.idl

```
struct TimeOfDay {
    short hour;   // 0 - 23
    short minute; // 0 - 59
    short second; // 0 - 60
};

interface Time {
    TimeOfDay get_gmt();
}
```

From IDL to Code



A Time Server


```
#include <iostream>
#include <time>
#include "timeS.hh"

class Time_impl: public virtual POA_Time {
public:
    virtual TimeOfDay get_gmt()
        throw(CORBA::SystemException) {
        struct tm *time_p = gmtime(time(0));

        TimeOfDay tod;
        tod.hour   = time_p->tm_hour;
        tod.minute = time_p->tm_min;
        tod.second = time_p->tm_sec;

        return tod;
    }
}
```


A Time Server (cont'd)

```
int main(int argc, char *argv[])
{
    // Start ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    // Create object
    Time_impl time_servant;

    // Issue OID
    Time_var tm = time_servant._this();
    cout << orb->object_to_string(tm) << endl;

    // Now go!
    orb->run();
}
```

A Time Client

time.idl

time.hh

```
#include <iostream>
#include "time.hh"

int main(int argc, char *argv[])
{
    // Start ORB
    CORBA::ORB_var orb = CORBA::ORB_init(argc, argv);

    // Lookup reference
    CORBA::Object_var obj =
        orb->string_to_object(argv[1]);
    Time_var tm = Time::_narrow(obj);

    // Access server
    TimeOfDay tod = tm->get_gmt();
    cout << "Time: "
         << tod.hour << ":" << tod.minute << endl;

    return 0;
}
```


Client and Server

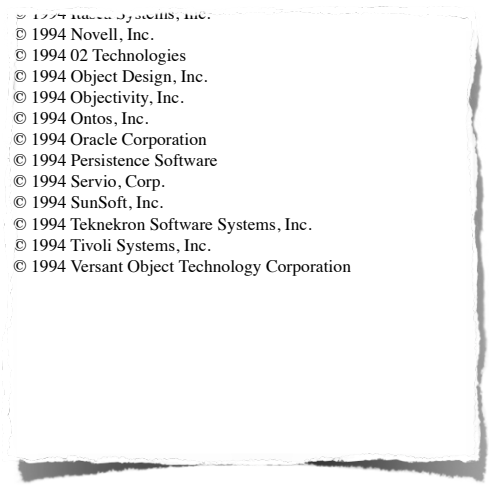
```
$ ./server &
corbaloc::134.96.235.32:38693/
StandardNS/NameServer-POA/_time
$ ./client corbaloc::134.96.235.32…
Time: 10:50
$
```


CORBA Services

- Name services
  map names to objects
- Persistence services
  for saving state
- Notification services
  objects register for events
- Security services
  authorization and encryption
- Life cycle services
  creating and moving objects
- Transaction services
  two-level commit

CORBA Services

The Problem with CORBA



http://en.wikipedia.org/wiki/Design_by_committee

Design by Committee

- CORBA was designed by a large committee of vendors
- No single master architect
- Standards created as a *union* of all proposals, with no regard to coherence
- Vendors not interested in customers moving between implementations

CORBA Consequences

- CORBA has lots of standards…
  (some of them with proprietary extensions)
- …but few implementations
  (Background: first came the specs, then the implementations)
- Still has a hard time being adopted today
- Implementations try to lock in users

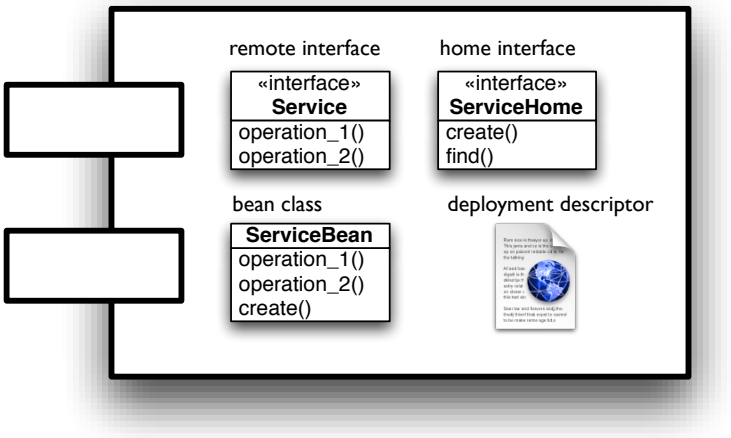
http://www.davidchappell.com/articles/article_Trouble_CORBA.html

http://en.wikipedia.org/wiki/Java_EE

Java EE

- Java EE (Java Platform Enterprise Edition)
- widely used platform for distributed architectures
  JBoss • IBM WebSphere • Apple WebObjects • Oracle Application Server • BEA WebLogic
- composed of *Enterprise Java Beans* (EJBs)

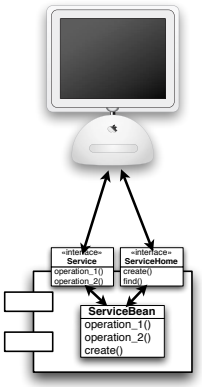
Enterprise Java Bean



Enterprise Java Bean

- *Remote interface* – a Java interface for invoking services
- *Home interface* – for creating EJBs and discovering services
- *Bean class* – implementation
- *Deployment descriptor* – describes services in XML

Enterprise Java Bean



- *Remote interface* – a Java interface for invoking services
- *Home interface* – for creating EJBs and discovering services
- *Bean class* – implementation
- *Deployment descriptor* – describes services in XML

Java EE

- Java EE provides several *services*
  like CORBA: clustering, security, transactions, persistence…
- Integrates nicely with Java language
  (no IDL, for instance)
- Good if application is 100% Java
- Similar: Microsoft .NET assemblies
  (in C# and other .NET languages, for instance)

Fallacies of Distributed Computing

1. The network is reliable.
2. Latency and transport cost are zero.
3. Bandwidth is infinite.
4. The network is secure.
5. Topology doesn't change.
6. There is one administrator.
7. The network is homogeneous.

The list of fallacies generally came about at Sun Microsystems.
See http://www.rgoarchitects.com/Files/fallacies.pdf

Web Services

- Web Service =
  "a software system designed to support interoperable Machine to Machine interaction over a network" (W3C)
- Based on three standards:
  SOAP • WSDL • UDDI

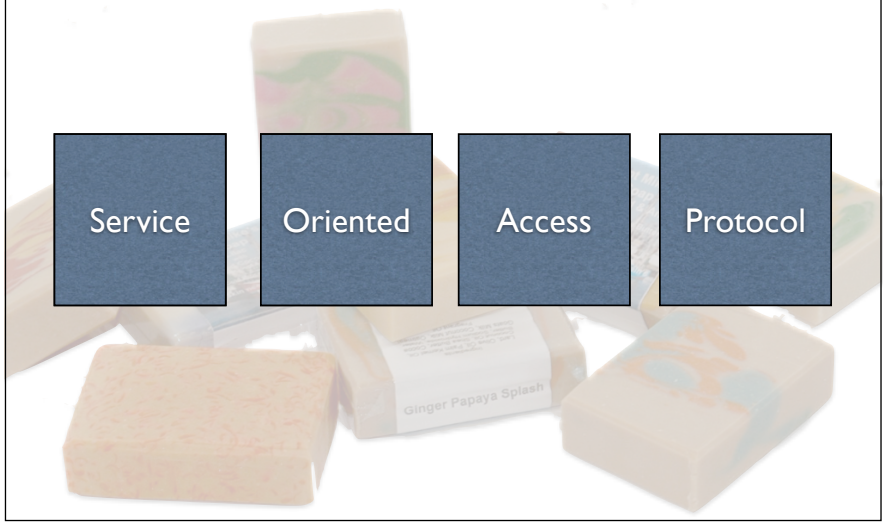
46

SOAP

"Simple Object Access Protocol": a pun on CORBA and others

47

SOAP



Originally "Simple Object Access Protocol": a pun on CORBA and others

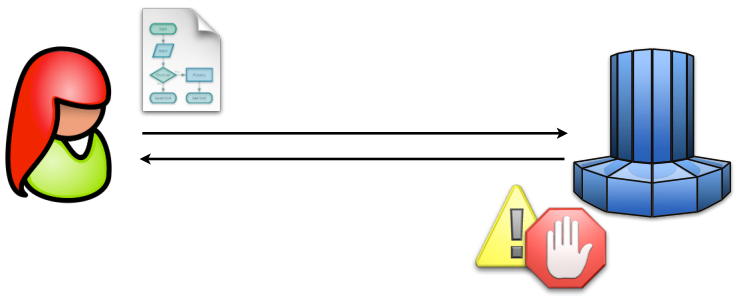
48

SOAP

SOAP

- Basic idea: *tunnel remote procedure calls* via another protocol (such as HTTP)
- All data is encoded in XML format

A Testing Server



A SOAP Query

The Header

```
POST /praktomat/check HTTP/1.1
Host: www.praktomat.org
Content-Type: text/plain
Content-Length: 1264
SOAPMethodName: urn:praktomat-org:CheckProcs#check
```

invoked method

A SOAP Query

The Body

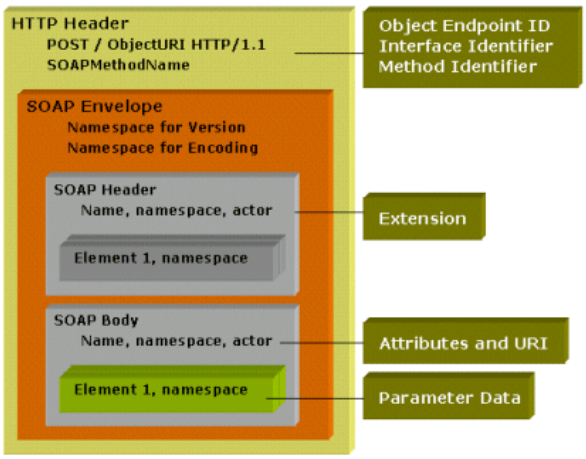
```
<Envelope>
  <Body>
    <cp:check xmlns:cp="urn:praktomat-org:CheckProcs">
      <theProgram>
      // My new program
      int main(int argc, char *argv[])
      …
      </theProgram>
    </cp:check>
  </Body>
</Envelope>
```

A SOAP Reply

```
<Envelope>
  <Body>
    <cp:checkResponse
        xmlns:cp="urn:praktomat-org:CheckProcs">
    <message_list>
      <message>
        <location>
          <file>main.cc</file>
          <line>45</line>
        </location>
        <text>undefined reference</text>
      </message> …
```

The reply can be translated into readable text

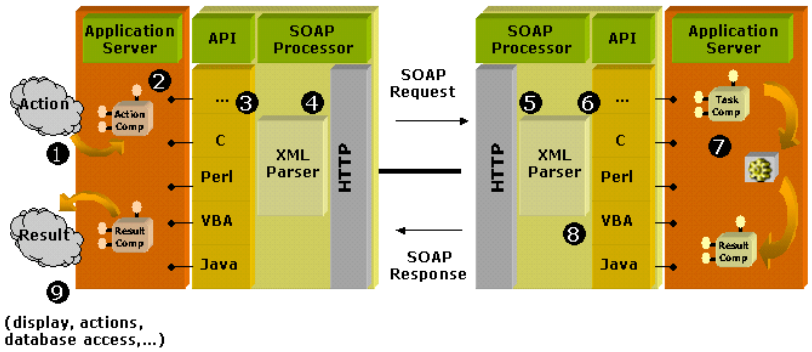
SOAP Messages



http://www.intranetjournal.com/articles/200012/id_12_13_00b.html

55

SOAP Communication



http://www.intranetjournal.com/articles/200012/id_12_13_00b.html

56

SOAP Communication

1. We invoke a service of the local application server…
2. …which uses a service via its API
3. The API uses SOAP to pack the data into XML…
4. and send it to the server via HTTP
5. The server parses the XML data
6. The message is routed to the appropriate service
7. The application executes the task
8. The return is done in the same way – packing, unpacking…
9. The result can be displayed, stored, processed…

1) Station 1 executes a command which creates an action on the associated application server.
2) This command generates a process within the application and the result arrives in the application interface.
3) The message is translated into XML format by the implementation and is then sent to the Web server.
4) The XML parser checks the coherence of the XML document and sends the SOAP message via HTTP.
5) The XML parser checks the validity of the message using the HTTP and XML headers, and accepts or rejects it.
6) The message is then routed to the relevant application server and translated by the implementation so

57

SOAP vs CORBA

- SOAP needs more bandwidth and more processing time than binary protocols
  communication 10–15 times more bandwidth,
  processing 30–60 times slower than IIOP in CORBA
- SOAP is easy to deploy and integrate
  especially across firewalls
- CORBA and other middleware can be built on top of SOAP

http://manoftoday.wordpress.com/2006/09/29/in-depth-investigation-of-corba-vs-soap/

WSDL

- WSDL = *Web Service Description Language*
- describes how a Web service is used
- similar to IDL, but in XML format

WSDL elements

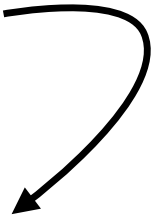
- types
  data type definitions used to describe the messages exchanged.
- message
  an abstract definition of the data being transmitted
- portType
  a set of abstract operations, refering to an input message and output messages
- binding
  concrete protocol and data format specifications for the operations and messages defined by a particular portType

WSDL elements

- port
  an address for a binding, thus defining a single communication endpoint
- service
  aggregate a set of related ports

A Banking Service

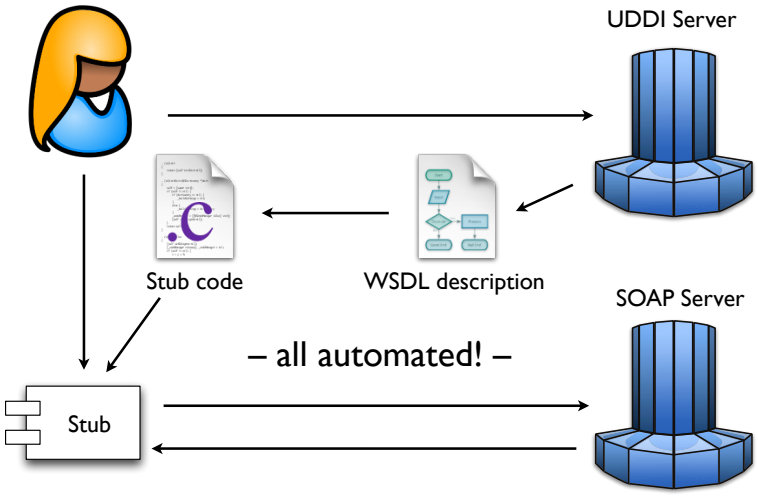
```
<element name="withdrawData">
  <complexType>
    <sequence>
      <element name="account" type="long"/>
      <element name="amount" type="float"/>
    </sequence>
   </complexType>
</element>

<element name="withdrawResponse">
  <complexType>
    <sequence>
      <element name="newBalance" type="float"/>
      <element name="amount" type="float"/>
      <element name="overdrawn" type="boolean"/>
    </sequence>
  </complexType>
</element>
```



UDDI

- UDDI =
  Universal Description Discovery and Integration
- Planned as a worldwide registry for Web services
- Mostly used as *corporate/private registries* within a company

All Combined

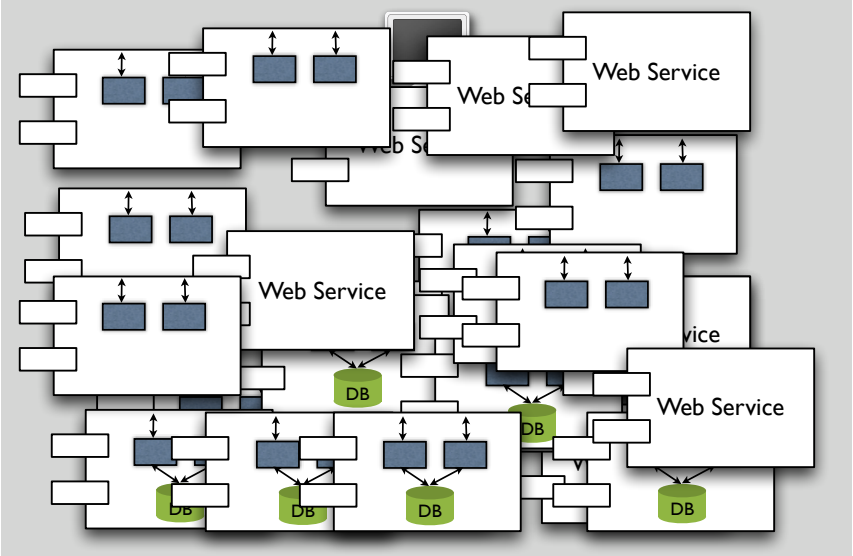


A programmer searches for a Web service with UDDI
The WSDL description is translated into IDL
generates a stub from the IDL
can use the Web Service via the stub as if it were available locally

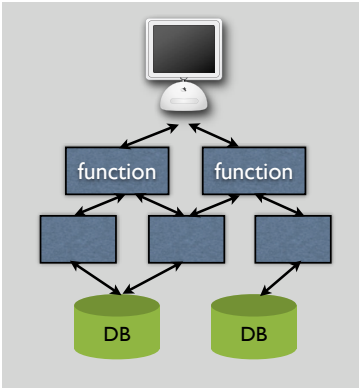
Service-Oriented Architectures

Goal – decompose systems into *components*

- …all with interfaces described by WDSL
  (or similar)
- …all composable with SOAP
  (or similar)
- loosely coupled along (Web) services
- interoperable across languages and platforms



Migrating to SOA

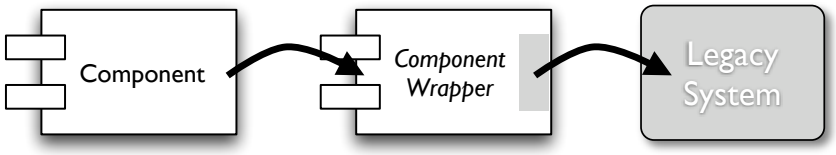


How do we migrate an existing system to components?

Wrappers

- Basic idea: wrapping a system (or parts thereof) behind *component interface*
- User does not notice that there is a legacy system behind the interface
- Used for *migration:* First wrap, then replace components by new implementations

Component Wrapper

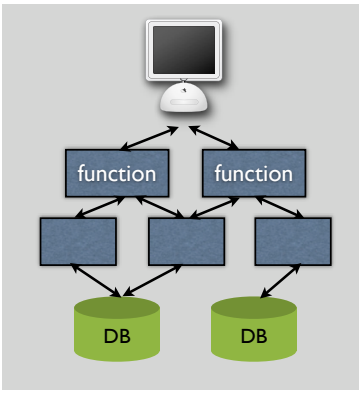


- Wrapper provides *component interface*
- Legacy system remains unchanged
- New components connect to wrapper

Procedural Wrapper

- Wrapper provides *procedural interface*
- Allows replacing parts of legacy system
- Legacy systems connect to new components

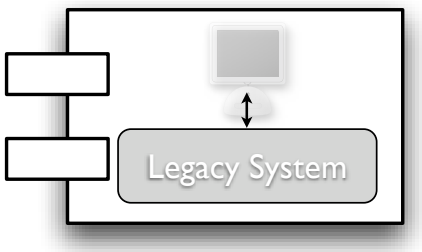
Migrating to SOA



How do we migrate an existing system to components?

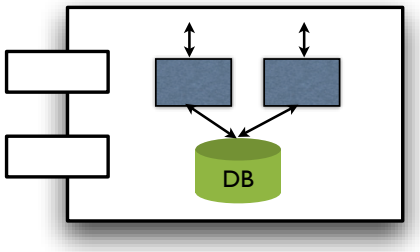
1. Simulate user I/O
2. Call procedures
3. Access data directly

Simulate User I/O



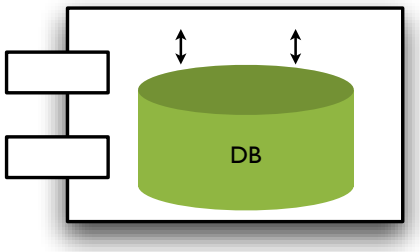
- The component simulates user interaction
- Problem: efficiency

Call Procedures



- The component calls individual procedures
- Problem:
  interference with other parts of the system

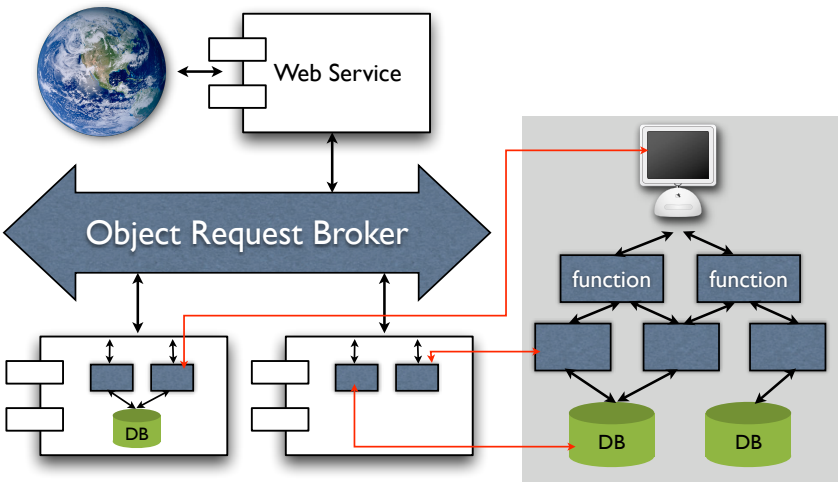
Access Data Directly



- The component manipulates data directly
- Problem: large (re-)implementation effort

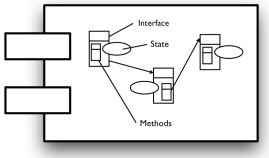
3. issue: implementation effort

Real Systems

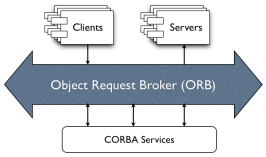


Real systems employ a multitude of techniques!

What's in a Component?

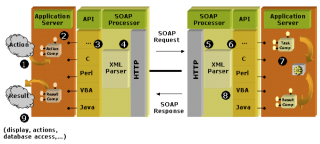


CORBA Architecture



Summary

SOAP Communication



Real Systems