

SOA – Patterns and Best Practices

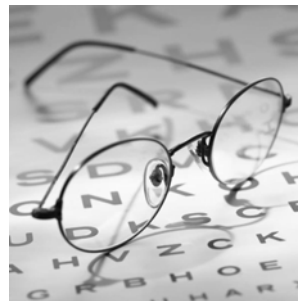
Michael Stal
Senior Principal Engineer
Siemens AG, Corporate Technology
Michael.Stal@siemens.com



Software &
Engineering
Architecture

Agenda

- SOA Motivation
- Implication of SOA Principles:
Patterns, and Best Practices
- Quality Related Best Practices
- Summary

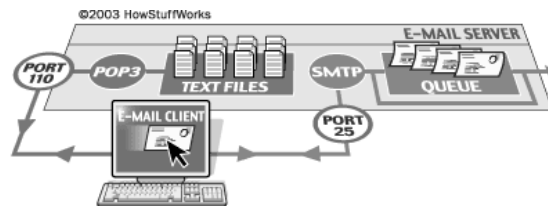


Software &
Engineering
Architecture

SOA is not Web Services - Example E-Mail

- **E-Mail denotes a typical example of a SOA architecture**

- Only one kind of service
- ... but offering a standard interface
- Communication using asynchronous message exchange
- Implementation-agnostic: different mail client and mail server implementations



© Siemens AG, CT SE 2, Michael Stal, 23.11.05
3



Software &
Engineering
Architecture

Tracking the SOA Principles



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
4

Interfaces and Schemas (1)

- **Role based interfaces**

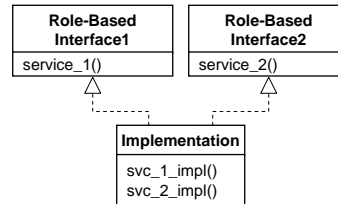
- Minimize impact of future changes

- **Explicit Interfaces**

- Contract with the interface not the implementation

- **Encapsulated Implementations**

- Hide implementation details



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
5

Interfaces and Schemas (2)

- **Explicit, Role-based Interfaces represent a Service Interface expressed using a Schema (e.g., XML)**

- **Schema contains structural information (service) as well as physical binding (network address)**

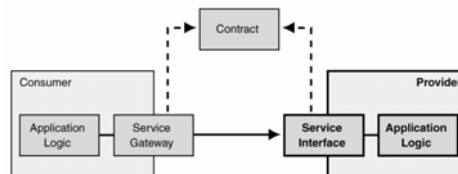
- **The Service itself (Implementation) is loosely bound to the service interface**

- **Useful to leverage generator techniques for providing infrastructural glue**

- **Consequences**

- Clients and services unaware of each other's implementation technology
 - Interface standardized instead of code
 - systems are heterogeneous => interface must stick to least common denominator

- **Examples: CORBA, DCOM, E-Mail**



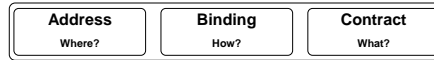
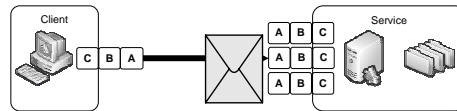
Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
6

ABC of a Service

- **A Service (Endpoint) contains 3 ingredients:**

- A: An Address that tells **where** the service resides
- B: A Binding that tells **how** to reach the service regarding protocols
- C: A contract that defines **what** the service provides in terms of operations



Endpoint

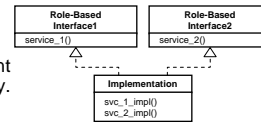


Software & Engineering Architecture

Best Practices

- **Extension Interface Pattern:**

- Problem: how to prevent interface bloating
- Solution: Group different responsibilities/roles in different interfaces. Consider to introduce navigation functionality.



- **Design by Contract (Contract First):**

- Problem: Preventing implementation dependencies
- Solution: When designing top-down rely on a contract-first approach
- Main Benefit: Role-Based, implicit interface design
- Consider Service and Operation Contracts (see WCF aka Indigo):
 - Service Contract (~ WSDL:PortType), Operation Contract (~WSDL:Operation)



Software & Engineering Architecture

Best Practices (cont'd)

• Implementation-First

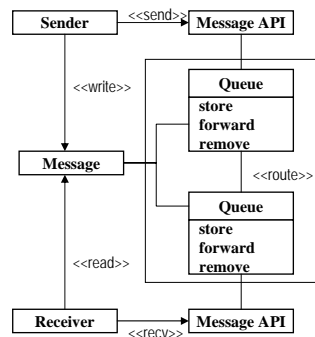
- Problem: How to integrate existing functionality
- Solution:
 - If implementation already existing figure out interfaces
 - If necessary, refactor and adapt legacy solution
 - Integrate service-oriented using existing interfaces
- Design and Implementation Issues:
 - Analyse kind of adaptation:
 - Is navigability bi-directional?
 - Are data types used compatible with SOA approach or are there further encapsulation layers necessary?



Software & Engineering Architecture

Message Exchange

- **Communication Paradigm: Asynchronous Message Exchange**
- **Stateless: no state preserved across message exchanges**
- **Message paths routed not fixed**
- **Binding can be delayed instead of being fixed**
- **Intermediaries might be involved**
- **Service Contract defines order of message exchange**
- **This approach can be used as a base for more advanced communication styles (eventing, publish/subscribe, multicast)**



Using Queues often not appropriate for embedded systems. You may leave them out!



Software & Engineering Architecture

Slide 10

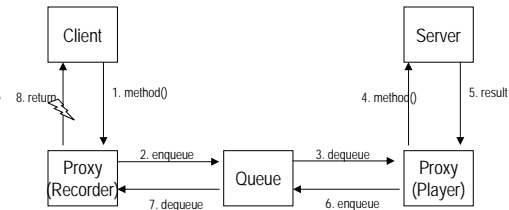
B18 Mention that queues are optional!

Buschmann_F; 14.09.2005

Broker Pattern: Synchronous Operation

- **Simulating Synchronous Communication possible by using**

- tokens to correlate different messages
- proxies to shield client and server from the asynchronous message protocol



© Siemens AG, CT SE 2, Michael Stal, 23.11.05
11



Software &
Engineering
Architecture

Best Practices

- **Think asynchronous, message-oriented and document-centric**

- Problem: Developers are used to synchronous communication style
- Solution: Even if synchronous layer is available make developers think in asynchronous message exchange
- Thus, a client does not call a service operation. It „send a document to a service“ and may or may not obtain an answer
- Example: Event-driven scenarios such as control systems, some business systems, network management do not work with synchronous RPCs

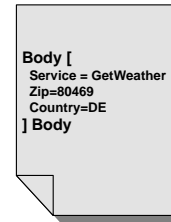


Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
12

Messages and Documents

- Each message transports a document
- A document must provide
 - mandatory content
 - optional context (e.g., QoS info)
- Sender and Receiver must agree on document content
 - Either by negotiating document structure and semantics individually
 - Or by relying on a predefined specification
- Documents can be binary or text-based
- Examples: TCP, IIOP, SOAP, MoM, E-Mail, Snail Mail

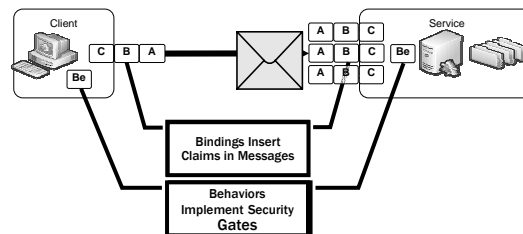


Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
13

Best Practices

- Consider Message- and Data Contract
 - Message Contract: what's in the header and what's in the envelope (~ WSDL:Message)
 - Data Contract (~XSD)
- Plan for Out-Of-The-Band Functionality/Policies
 - For Out-Of-the-Band Information design headers and interceptors



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
14

Best Practices (cont'd)

- Define message-exchange patterns to use:

- 1:1 Communication:
 - Oneway
 - Callback
 - Request/reply
- n:m Communication
- This depends on domain: e.g. Event driven communication uses n:m style



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
15

SOA Processes

- We get the SOAdvantage not by isolated invocation of single services
- But by the orchestration and composition of services to processes
- Using a Domain Specific Language (DSL) a process description might be compiled into a service of its own

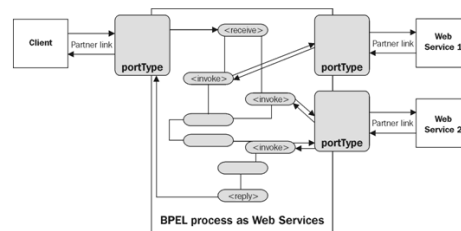


Figure: Example BPEL process



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
16



Software &
Engineering
Architecture

Best Practices

- **Business First Approach:**

- Derive functional services from your business case / domain. Classify services, e.g. Entity/Resource services, Infrastructure Services, Session Services. **Session Services are the entry point. No direct access to entities or other implementation details using services!!!!**
- If existing components are to be integrated design service facades
- Derive work flows from use cases. If necessary introduce specific workflow DSL or use BPEL
- Introduce strategic views: security domains, business domains, management domains, error handling domains
- Add views for operational qualities: e.g., fault tolerance/scalability clusters. Decide which use cases have specific requirements such as transaction support
- Developmental qualities: Use Commonality/Variability Analysis to identify change cases and use rules engines to support variability
- Build service mocks to test your system but always test against real services if possible



© Siemens AG, CT SE 2, Michael Stal, 23.11.05
17



Software &
Engineering
Architecture

Beyond the Basics – The Missing Pieces

- **In contrast to common belief, there is more to SOA than just using „low-end“ protocols such as SOAP, WSDL and UDDI**
- **We need complementary technologies for**
 - Semantic Integration
 - Quality of Service
 - Structural Integrity
 - ... name your own



© Siemens AG, CT SE 2, Michael Stal, 23.11.05
18

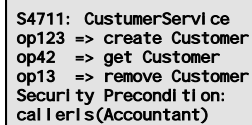
Semantic Integration (1)

- Here is a service interface
- It is only defined as a soup of structural information
- Now, guess, what the service actually provides to you?
- Semantics specifies the meaning
- Annotations are used to associate structure with semantics

```
Service S4711 {
  Int   op123(String)
  String op42(Int)
  void  op13(Int)
}
```



```
S4711: CustomerService
op123 => create Customer
op42  => get Customer
op13  => remove Customer
Security Precondition:
callerIs(Accountant)
```



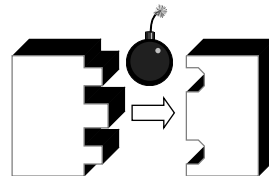
© Siemens AG, CT SE 2, Michael Stal, 23.11.05
19



Software &
Engineering
Architecture

Semantic Integration (2)

- Semantic Integration is also useful for
 - providing automatic service composition
 - on demand service interface adaptation based on ontologies
 - location of services using ontologies instead of full text searches
 - supporting of Quality of Service and SLAs by provisioning of meta information



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
20

Decentralized Approach

- **In the current approach the discovery of services**

- Either, is not necessary because all client/service connections can be statically configured
- Or, uses a central repository where all services are registered



- **In the future**

- Dynamic discovery using Peer-To-Peer and PnP technologies might be a better strategy at least for some parts of the system
- A compromise might be to use a hybrid approach: decentralized access where possible and centralized access where necessary

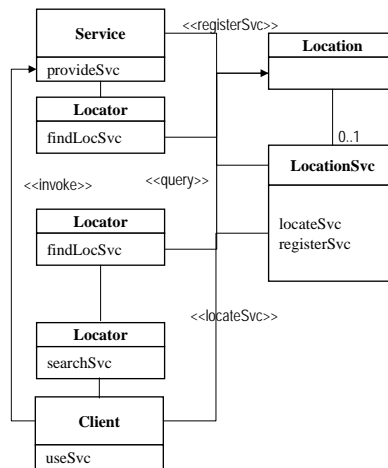


Software & Engineering Architecture

Location Pattern

- **Best Practices:**

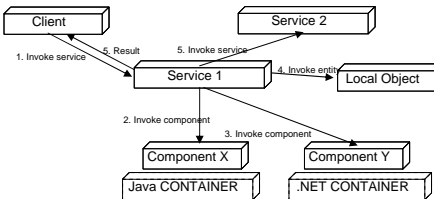
- For clients provide indirection layer to hide details of how locations are found
- If decentralized approach is required, use Lookup/Location patterns



Software & Engineering Architecture

Structural Integrity

- Today, reality provides a mixture of different paradigms:



- **Common Approach: SOA for external integration and other technologies such as OO middleware/componentware for internal integration**
- **Liability:**
 - Architectural Drift is inevitable. Systems become brittle and instable
 - Architects and developers must now deal with standard platforms AND with SOA
 - Integration often becomes a nightmare because of sheer complexity and size



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
23

Service Components and Enterprise Service Bus

- **Basic Idea (One Size Fits All):**
 - Shield developers from technology and enable developers to use the SOA paradigm for internal and external integration
 - Different technologies can be plugged-in transparently (Open/Close Principle)
- **Best practice: Use ESB/Container pattern**
- **This implementation should be provided by infrastructure provider**
- **Examples: Microsoft WCF (Windows Communication Foundation), IBM Websphere**



Software &
Engineering
Architecture

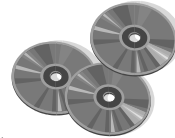
© Siemens AG, CT SE 2, Michael Stal, 23.11.05
24



Software &
Engineering
Architecture

Stateful Services

- **In the pure SOA approach clients, services and connections are stateless.**
 - Stateless is good because it fosters scalability and flexibility
 - Stateless is evil when we need to manage sessions (i.e., correlated invocations) or when service affinity is important
- **Different options**
 - Implementing Sessions:
 - State is always transferred between peers. Network load may become an issue!
 - Send a token between peers and let one of the peers (either client or service) maintain state in a store using the token as primary key
 - Guaranteeing affinity
 - Include context information in messages that tell infrastructure how to dispatch the message
 - Could be done manually on application layer (bad idea due to incompatibility) or on infrastructure layer (good because hidden from developers)



© Siemens AG, CT SE 2, Michael Stal, 23.11.05
25



Software &
Engineering
Architecture

Quality Related Principles and Patterns: Service Design (1)

- Each client call to a service is potentially a remote call that is significantly slower than a local method call, due to networking overhead, latency, and jitter
- Each service call can involve different activities with hidden costs (security, activation / passivation, ...)
- Naïve service interface design and use can cause significant performance and scalability penalties.



© Siemens AG, CT SE 2, Michael Stal, 23.11.05
26

Quality Related Principles and Patterns: Service Design (2)

Example for a service interface that is not recommended because the client needs to make many fine-grained calls execute the service:

```
service MyFlightBookingService {
    FlightList searchFlight(String departureAirport,
                          String destinationAirport,
                          ...);
    void selectFlight(String flightId);
    void setTravellerFirstName(String firstName);
    void setTravellerLastName(String lastName);
    void setPersonId(String id);
    ...
}
```

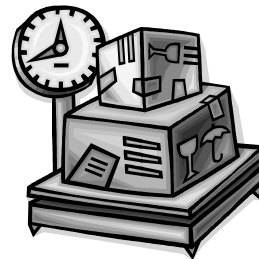


Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
27

Quality Related Principles and Patterns: Service Design (3)

- Only realize course-grained, self-contained business processes as services
- Let services manage their workflow via Session Facades that use local components for further processing
- Use Transfer Objects and Value List Handlers to increase the amount of data passed between clients and a service
- Never expose service attributes via getter/setter methods
- Loose coupling is key: let clients access a service via a Service Locator and access its functionality via Business Delegates. Access persistent data via Data Access Objects

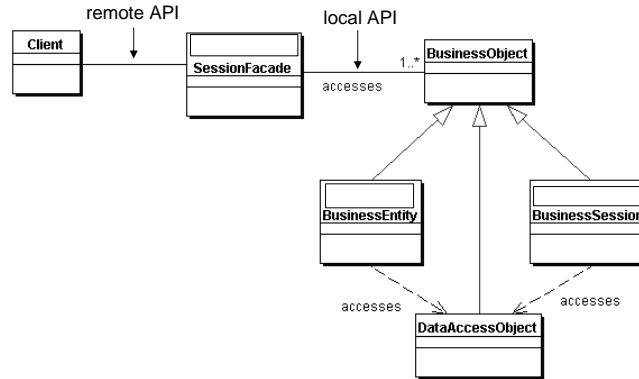


Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
28

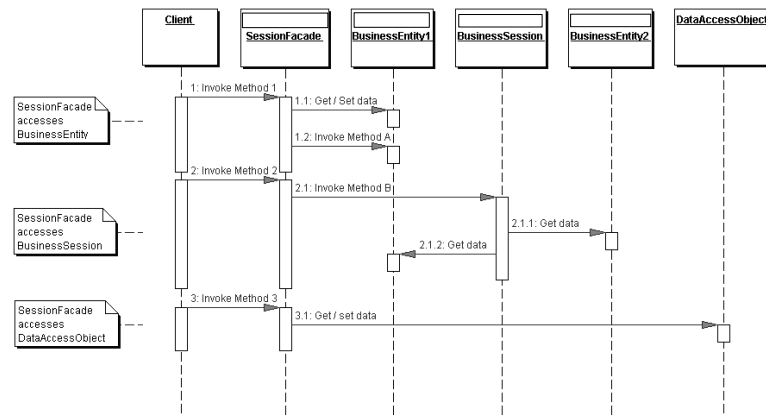
Session Facade (1)

- Encapsulates the complexity of interactions between the business objects
- Provides a uniform coarse-grained service access layer to clients



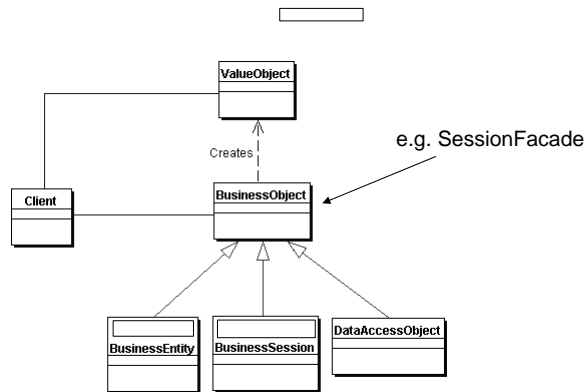
Software & Engineering Architecture

Session Facade (2)



Software & Engineering Architecture

Transfer Object



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
31

Large Result Lists

- Clients might want perform searches or obtain large result lists from services
- Search methods have a huge performance overhead when used to perform large searches
- Typically, a client uses the results of a query for read-only purposes, such as displaying the result list.
- Often, the client views only the first few matching records, and then may discard the remaining records and attempt a new query

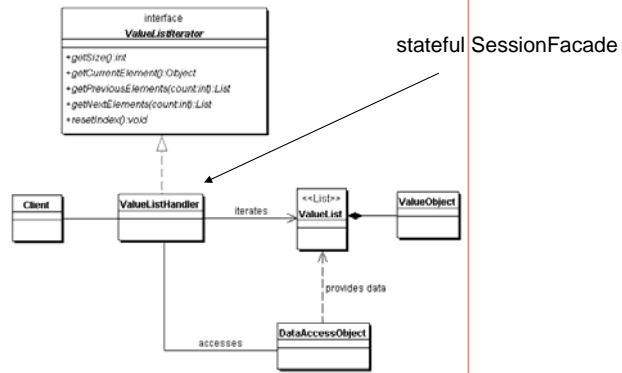


Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
32

Value List Handler

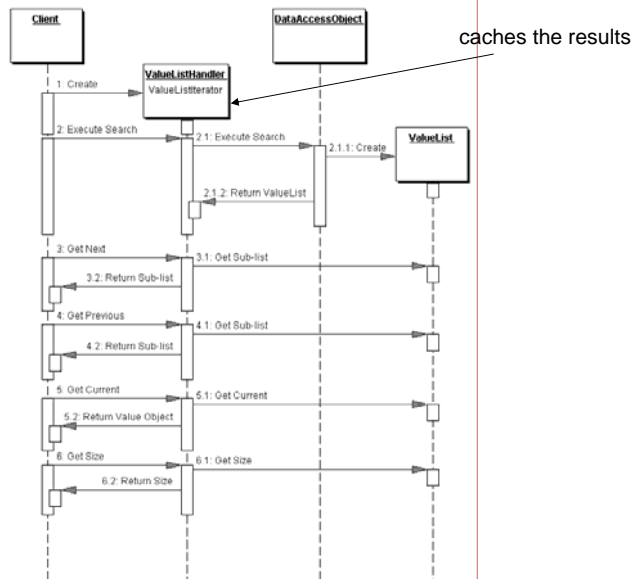
Use a Value List Handler (aka Batch Method) to search, cache the results, and allow the client to traverse and select items from the results:



© Siemens AG, CT SE 2, Michael Stal, 23.11.05
33



Software & Engineering Architecture



© Siemens AG, CT SE 2, Michael Stal, 23.11.05
34



Software & Engineering Architecture

Loose Coupling

Client should not interact directly with a provided service interface:

- API changes affect client code
- tight coupling to all—even unused service methods—decreases system flexibility
- client must handle communication details
- the client is not transparent to discovering services

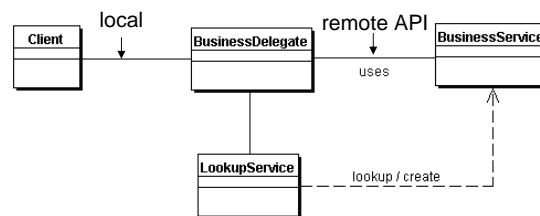


Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
35

Business Delegate

Use one or more role-based Business Delegates (Extension Interfaces) to reduce coupling between clients and services. Each Business Delegate hides underlying implementation details of a remote service, such as lookup, dependencies to the provided service interface, and remote communication details.



Often, business delegates are developed by the service developers and provided to the client developers.

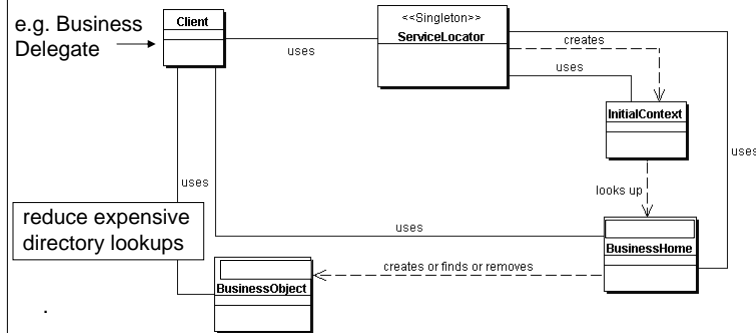


Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
36

Service Locator / Lookup Service

Use a Service Locator object to abstract all repository usage or service discovery and to hide the complexities of initial creation.



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
37

Data Access

- Most enterprise applications use relational database management.
- Data can reside in many types of repository:
 - mainframe systems
 - Lightweight Directory Access Protocol (LDAP) repositories
 - flat files
 - object-oriented databases
 - business to business systems
- Services may access a data store or EIS System to manage persistent data

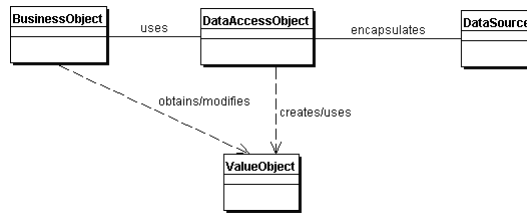


Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
38

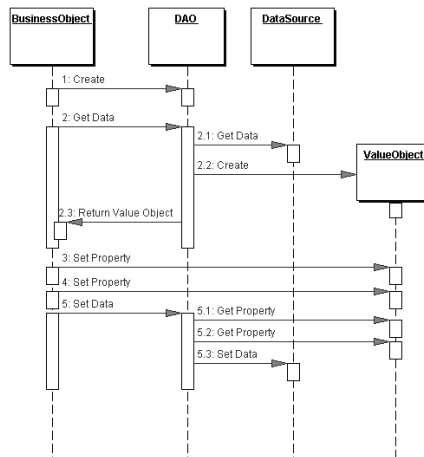
Data Access Object (1)

Use data access objects to encapsulate access to the data source and to achieve portability:



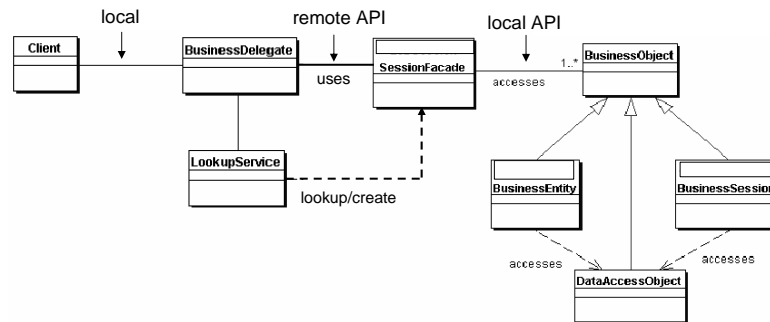
Software & Engineering Architecture

Data Access Object (2)



Software & Engineering Architecture

Proposed Service Architecture



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
41

Summary - SOA from 10000 feet

- SOA has a strict business-oriented view: services are self-contained and provide a business value.
- Loose coupling is the SOA mantra!
- Architecture is key: many patterns support SOA!
- Thinking SOA is not trivial ...



- asynchronous communication, asynchronous service implementations.
- inversion of control, dependency injection
- strict design by contract
- operational qualities

... but if done consequently and disciplined helps building platforms and product-lines for flexible, high-quality systems.

- SOA is an *additional* mindset for building software, not a replacement of existing approaches: components, objects, and other paradigms still have their place.



Software &
Engineering
Architecture

© Siemens AG, CT SE 2, Michael Stal, 23.11.05
42