CSE 210: Service-Oriented Software and Systems Engineering & Model-Driven Development

Ingolf H. Krueger
ikrueger@ucsd.edu, http://sosa.ucsd.edu

with contributions from Roshni Malani, Michael Meisinger, Massimiliano Menarini, Stephen Pasco

Department of Computer Science & Engineering
University of California, San Diego
La Jolla, CA 92093-0114, USA

California Institute for Telecommunications and Information Technologies
La Jolla, CA 92093-0405, USA
Quote of the Day

“To manage a system effectively, you might focus on the interactions of the parts rather than their behavior taken separately.”

-- Russell L. Ackoff
Overview

• Introduction and Motivation
• Practical Applications of Services
• Service-Oriented Architectures
• Benefits of Service-Oriented Approach
• Methodology
• Tool Support
• Summary and Outlook
Examples

• Consider Google™
  - Provides search, cache, and spelling services
  - Accessible via Internet
  - Across different client platforms
  - Can be integrated into your application
  - Can use while developing in any environment

• Consider Amazon.com®
  - Interacts with external services
  - Provide many more complex services
    • Access to product data, content from customers, seller information, and shopping carts
  - Allows third-party vendors to sell on site

• Consider Central Locking System
Central Locking System – Crosscutting Services

operation of locks / signaling

fetch driver presets on entry

crash detection / management

set tuner presets
Motivation

• Increasing size and complexity
  – Complexity derives from
    • Component distribution
    • Component interaction
  – Need a way to model, design and implement complex interactions

• Constant change and evolution
  – Market needs drive change
  – Changes often span across the entire system
  – Need a way to capture functionality that cross-cuts the system

• Convergence between business and technical systems

• Integration and interoperability of independently designed and implemented components
Service Notions

- Telecommunications
  - SAPs
  - Features
- Client/Server Architectures
  - Server Component APIs
- Embedded Systems
  - Function Networks
- Web Services
  - Functions accessible via Internet
  - Registration/Discovery/Use
  - Lightweight, standardized transport/access
- Other Influences
  - Use cases
  - Aspect-oriented programming
Finding a Service Definition

- All service notions are based on component interaction

- Key Challenges:
  - Expression of partial interactions
  - Composition of services
  - Service refinement
  - Verification and validation
Service:
Interaction pattern required to accomplish a specific task.

- Compatible with/generalization of existing definitions
- Addresses cross-cutting nature of services
- Provides abstraction level independent of “component”
- Applicable across domains
Overview

- Introduction and Motivation
- Practical Applications of Services
- Service-Oriented Architectures
- Benefits of Service-Oriented Approach
- Methodology
- Tool Support
- Summary and Outlook
Service-Oriented Architectures in Practice

• Service-Orientation in different application domains
  – Automotive Systems, Embedded Systems
  – Telecommunications Systems
  – Networking
  – Business Information Systems, Web services
  – Mobile Applications
  – Sensor Networks

• Service-Oriented Architectures
  – Web service Middlewares
  – Service Lookup

• Services as high level view on systems
  – Structure sets of related domain objects
Application Domain: Automotive systems

- Embedded, reactive controllers, for
  - Central locking system
  - Crash management
  - Power windows & seats
  - Multimedia system
  - Navigation system
  - Airbag controllers
  - etc.
Application Domain: Automotive systems

• Problems:
  – Exponentially increasing complexity of systems and development
  – Interactions between distributed components are causing complexity
  – Current development approaches leave it difficult to address system-wide functions and quality concerns properly
  – Specifying separate components completely and integrating them is very difficult
Application Domain: Automotive systems

- **Benefits of Service-Orientation:**
  - Service-oriented approach allows separation of the different vehicle functions from technical deployment units
  - Service-oriented middlewares increase standardization and abstract from technical entities
  - Development for product lines is facilitated by providing reusable, independent functions

- A service-oriented approach is also applicable for other embedded systems and controllers
Application Domain: Telecommunications Systems

- Telecommunication systems
  - Network switches
  - Telephony & data networks
  - Mobile phones
Application Domain: Telecommunications Systems

• Facts and Problems:
  - Extremely large, complex distributed systems
  - Heterogeneous deployments
  - Features drive innovation and are the increments of design, development and testing
  - Undesirable feature interactions block progress; features cannot be developed independently anymore
Application Domain: Telecommunications Systems

• Benefits of service-oriented approach
  – Provide a domain-independent view on telecommunication features
  – Services can be more abstract than telecommunications features or telecommunications services
  – Service-orientation extends and advances the ideas of feature-oriented development
  – Abstracts from technical solutions and infrastructures
  – Puts interactions in the center of concern
  – Allows detection of feature interactions on the model level
Application Domain: Business Information Systems

- **Applications**
  - Multi-tier web applications
  - Layered business information systems
  - Web service infrastructures
  - Enterprise Information Systems
  - Systems Integration

Service Oriented Middleware
Application Domain: Business Information Systems

- **Problems and Facts:**
  - Business Information Systems get very complex
  - Number of distributed components increases
  - Several tiers within software architectures
  - Complex dependencies between system components and complex interactions
  - Difficult to understand, model and verify systems completely
  - Structuring systems based on layers or functional units does not show the system functions anymore
Application Domain: Business Information Systems

• Advantages of Service-Orientation
  – “Web services” as well-established technology
    • Emerging protocols and standards (SOAP, UDDI, WSDL, …)
    • Industry-wide attention and application
    • Compatibility between middlewares
  – Services exhibit clearly defined system functions to the environment
  – Services provide access points in a distributed environment
  – Services allow loose coupling of components using open-standard, light-weight communication protocols
  – Services can be registered and looked up using service description languages and registries
Application Domain: Mobile Applications

- Applications
  - Context-aware or context-adaptive services
  - Location-based services
  - Ad hoc networks
  - Service lookup in unknown networks
Application Domain: Mobile Applications

• Problems
  - Highly heterogeneous, distributed, resource-limited networks
  - Users expect functionality in form of usable functions or services
  - Users expect services to consider the current location and environment
  - The execution environment is unspecified and can change significantly
Application Domain: Networking

• Networking Applications
  - Protocol Stacks
  - Service Access Points
  - Quality-of-Service

• Standards
  - ISO/OSI
  - TCP/IP
  - TINA-C
  - IN
Application Domain: Networking

- **Difficulties**
  - Precisely define service interfaces and service access points
  - Precisely define Quality of Service
  - Consider layering, and virtual services (connections) provided by layers
Application Domain: Sensor Networks

- **Sensor network**
  - distributed system of sensing resources
  - electronic sensors of multiple types
  - networks can be hierarchical, ad hoc, mobile, etc.
  - apply data fusion algorithms

- **Sensor network middleware**
  - Provides abstraction between applications and sensor network implementation
  - Allows application to adapt to changes in the underlying sensor network (e.g., failure, hardware updates).
Application Domain: Sensor Networks

• Drawbacks to traditional approach to sensor application development:
  – Very low-level programming
  – Sensors addressed statically and uniquely
  – Program uses raw data types
  – Reliable communication protocols rarely supported

• Problems:
  – NOT Scalable
  – NOT Flexible to change
  – NOT Portable
Application Domain: Sensor Networks

- Service-orientation in sensor networks provides
  - Scalability
    - Distributed, Service-Oriented Architecture
    - Composition and Federation Patterns
    - Stream and Channel Patterns
  - Flexibility to Change
    - Location Transparency
    - Reliable Service Invocation
  - Portability
    - Adapter pattern
Application Domain: Sensor Networks

- Further benefits
  - Composability
    - Composite sensors
  - Hardware abstraction
    - Sensor adapter
  - Location transparency
    - Lookup service
    - Robustness through redundancy
  - Data typing & Adaptive communication
    - Sensor data stream
    - Channel
Overview

- Introduction and Motivation
- Practical Applications of Services
- Service-Oriented Architectures
- Benefits of Service-Oriented Approach
- Methodology
- Tool Support
- Summary and Outlook
Service-Oriented Architectures

• Service-Oriented Architecture
  - Multiple varying definitions exist
  - Collection of (network-accessible) services, each addressing a business function
  - Enables service interaction
    • by simple data passing
    • as several services coordinating some activity
  - provides means for registering of and connecting to services

• Service
  - Orchestration/Coordination of domain objects to deliver (business) function
  - Described as interactions among roles
  - Internal vs. external services
Service-Oriented Architectures

• Middlewares
  - CORBA ORBs (Object Request Brokers)
  - DCOM
  - Web Service Middlewares
    • J2EE
    • .NET

• Service infrastructure
  - Service registration and lookup
  - Communication using standard protocols and formats
Service-Oriented Architectures

Definition of

• Services that can be called upon
• Syntactic interface
• Discovery/Access mechanism
• Transport protocol
Web Services – Technologies

Web Service Description Language (WSDL)

Simple Object Access Protocol (SOAP)

Universal Description, Discovery, and Integration (UDDI)

Syntactic Interface ("IDL")

Transport Protocol

Registration/Discovery
Web Services Architecture*

- Security
- Transactions
- Reliable Messaging
- Metadata
- Messaging
- XML and SOAP
- Messaging Transports
  HTTP, HTTP/S, TCP, UDP, SMTP, ...

Service Registration and Lookup

• Motivation for a Service Registry
  - Service providers register their service at the registry
  - Service requesters can lookup at runtime services they need
  - Service registry matches service requesters and available services, using a service descriptor

• Motivation for a “Web Service Descriptor”
  - Each service provider could implement the Web Service differently
WSDL

• Describes a service, its methods and where it can be found
• How are parameters encoded (Encoded or Literal)
• How is the SOAP Body element encoded
  (RPC or Document)
• Can be used to (automatically) build a proxy
Sample WSDL Document

<definitions name="BabelFishService" lns:tns="http://www.xmethods.net/sd/BabelFishService.wsdl"
<message name="BabelFishRequest">
<part name="translationmode" type="xsd:string" />
<part name="sourcedata" type="xsd:string" />
</message>
<message name="BabelFishResponse">
<part name="return" type="xsd:string" />
</message>
<portType name="BabelFishPortType">
<operation name="BabelFish">
<input message="tns:BabelFishRequest" />
<output message="tns:BabelFishResponse" />
</operation>
</portType>
<binding name="BabelFishBinding" type="tns:BabelFishPortType">
<soap:binding style="rpc" transport="http://schemas.xmlsoap.org/soap/http" />
<operation name="BabelFish">
<soap:operation soapAction="urn:xmethodsBabelFish#BabelFish" />
<input>
<soap:body use="encoded" namespace="urn:xmethodsBabelFish"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</input>
<output>
<soap:body use="encoded" namespace="urn:xmethodsBabelFish"
    encodingStyle="http://schemas.xmlsoap.org/soap/encoding/" />
</output>
</operation>
</binding>
<service name="BabelFishService">
<port name="BabelFishPort" binding="tns:BabelFishBinding">
<soap:address location="http://services.xmethods.net:80/perl/soaplite.cgi" />
</port>
</service>
</definitions>
SOAP

• HTTP with an XML payload

• A SOAP Envelope contains an optional header and a body

• Format interoperability issues
Sample SOAP Message

POST /SimpleMathservice/Math.asmx HTTP/1.1
User-Agent: Mozilla/4.0 (compatible; MSIE 6.0; MS Web Services Client Protocol 1.0.3705.0)
Content-Type: text/xml; charset=utf-8
SOAPAction: "http://myadvancedwebserviceURI/SetSomeProperty"
Content-Length: 338
Expect: 100-continue
Connection: Keep-Alive
Host: localhost
<?xml version="1.0" encoding="utf-8"?>
<soap:Envelope
   xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xmlns:xsd="http://www.w3.org/2001/XMLSchema">
   <soap:Body>
      <SetSomeProperty xmlns="http://myadvancedwebserviceURI/">
         <str>SomePropertyValuesentfromclient</str>
      </SetSomeProperty>
   </soap:Body>
</soap:Envelope>
Web Services – Emerging Standards and Technologies

- WS-Coordination
- WS-Transaction
- Business Process Execution Language for Web Services (BPEL4WS)
- WS-Reliable Messaging
- WS-Addressing
- WS-Policy
- WS-Policy Assertions
- WS-Policy Attachments
- WS-Attachments
- SOAP with Attachments
- WS-Security Framework
- ...

April 21, 2005 © Ingolf H. Krueger

CSE
Overview

- Introduction and Motivation
- Practical Applications of Services
- Service-Oriented Architectures
- Benefits of Service-Oriented Approach
- Methodology
- Tool Support
- Summary and Outlook
Benefits of SOA

- Partial interaction specification
- Agility
- Flexibility
- Scalability
- Portability
- Encapsulation
- Abstraction
- Composability
Overview

• Introduction and Motivation
• Practical Applications of Services
• Service-Oriented Architectures
• Benefits of Service-Oriented Approach
  • Methodology
• Tool Support
• Summary and Outlook
Service Engineering for Distributed, Reactive Systems

Service-Oriented Applications

Tailored Development Processes

Service-Oriented Software Architectures

Expressive Description Techniques

Precise, Methodological Foundation

Tool Support
S3EL Methodology

• We address the full lifecycle of complex distributed reactive systems

• Focus on interaction from the beginning (services)
  – Describe interaction scenarios using MSCs
  – Describe relationships between scenarios with HMSCs
  – Specify QoS properties on interactions

• Abstraction from deployment
  – Use of roles as abstraction of interacting components
  – Mapping from roles to physical components is the last step of design
  – Technique to evaluate different deployment options

• Verify system properties on the service level

• Generation of code
  – As a base skeleton for implementation
  – To verify real time properties
Services in Domain Modeling

• Problem
  – Operation cannot be located within specific domain objects
  – Operations cross-cut responsibility of multiple objects/systems

• Solution:
  – Provide service layer(s)
  – Services are interfaces that stand alone in the model, without encapsulating state
  – Services orchestrate the interplay of multiple objects
Iterative development process

- Use cases captured as UML use-case diagrams or textual user stories
- Roles elicited out of use cases
- Services described as interaction between Roles using MSCs and HMSCs
- Domain model described in term of Roles
- Implementation architecture achieved mapping Roles to components.
- ADL provided to describe the mapping
- Different implementation architectures can be easily evaluated
Iterative development process

- **Service elicitation**
  - Use Cases
    - relationships expressed as use case graphs
  - Roles
    - many-one mapping to components
    - map to predicates over state space
  - Services
    - associate roles with interaction patterns
    - build up service repository
    - structural relationship expressed in a role domain model

- **Architecture definition**
  - Component Architectures
    - map roles to components (many-one)
    - component configuration expressed in a deployment domain model
  - Service Refinement and Refactoring
    - iteration with feedback
Notation

- Requirements captured and described using use case diagrams or user stories

- Services captured as interactions between Roles using MSCs (Message Sequence Charts)
Notation

- Interactions between services captured using HMSCs (High Level MSCs)

- Components and communication channels between described using component diagrams
Notation

- Communication channels modeled as streams

- Executable specification generated as state machines
Benefits

• Simple and intuitive graphical notation
  – Helps engineers to be more productive

• Precise semantic and solid mathematical foundation
  – Allows for formal verification of properties

• Iterative process and early milestones
  – Allows for short time to market and accommodates changing requirements

• Independence of service specification form deployment architecture
  – Allows for easy deployment of product lines and evaluation of different implementation architectures

• Code generation
  – Allows for fast development of systems where interactions are already taken care of and verification of QoS properties
Architecture Evaluation Approach

- Uses aspect-oriented techniques
- Easily evaluates different hardware deployments
Overview

- Introduction and Motivation
- Practical Applications of Services
- Service-Oriented Architectures
- Benefits of Service-Oriented Approach
- Methodology
- Tool Support
- Summary and Outlook
Tool Support

**S3EL-developed Tools**

- **M2Code**
  - Modeling tool

- **RTCGen**
  - Generate executable specifications based on RT CORBA
  - Validation of QoS properties

- **MSCCheck**
  - Verify design and implementation via model-checking

**External Tools**

- **Autofocus**
  - Tool from TUM used for verification
Interaction Model – Crucial for Model-Driven Development
M2Code

- Tool used to specify MSCs
- Integrated with MS Visio
- Generates state machines out of MSCs
- Produces XML based input files for all other tools
M2Code
RTCGen

- Generates code out of XML file exported form M2Code
- Currently generates RT CORBA based C code
- Code generation is template based: output language and platform can be easily changed
- Used to check QoS deadlines at runtime
• Verifies implementation against specification by model checking
• Verifies specification properties
• Integrated with the Eclipse framework
• Presents counterexamples as MSCs
Overview

- Introduction and Motivation
- Practical Applications of Services
- Service-Oriented Architectures
- Benefits of Service-Oriented Approach
- Methodology
- Tool Support
- Summary and Outlook
Summary and Outlook

• Concept of ‘services’:
  - Widely used
  - Not widely understood, yet

• Service:
  - Pattern of interaction among a set of roles

• Service-Oriented Architecture:
  - Modeling services as first-class entities

• Problems addressed:
  - Complexity, evolution

• Benefits provided:
  - Flexibility, scalability, portability, abstraction