Service-Oriented Architecture (SOA) and Software Product Lines: Pre-Implementation Decisions

Dennis B. Smith
dbs@sei.cmu.edu

Workshop on Service-Oriented Architectures and Software Product Lines (SOAPL)
13th International Software Product Line Conference (SPLC)
San Francisco, Ca
August 25, 2009

Software Engineering Institute | Carnegie Mellon

Agenda

Intersection of software product lines and SOA 

Selected practice areas and decision points

Variation mechanisms

Conclusions and next steps

Software Product Lines

Software product line

- a set of software-intensive systems that share a common, managed set of features satisfying the specific needs of a particular market segment or mission and that are developed from a common set of core assets in a prescribed way

Successful products lines enable organizations to capitalize on systematic reuse to achieve business goals

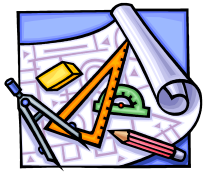
Benefits of software product lines

- productivity gains
- decreased development costs
- improved time to market
- higher reliability
- competitive advantage

Service Oriented Architecture

Service-oriented architecture (SOA) is a way of designing, developing, deploying and managing systems, in which

- Services provide reusable business functionality via well-defined interfaces.
- Service consumers are built using functionality from available services.
- Service interface definitions are first-class artifacts.
  - There is a clear separation between service interface and service implementation
- An SOA infrastructure enables discovery, composition, and invocation of services.
- Protocols are predominantly, but not exclusively, message-based document exchanges.



Services

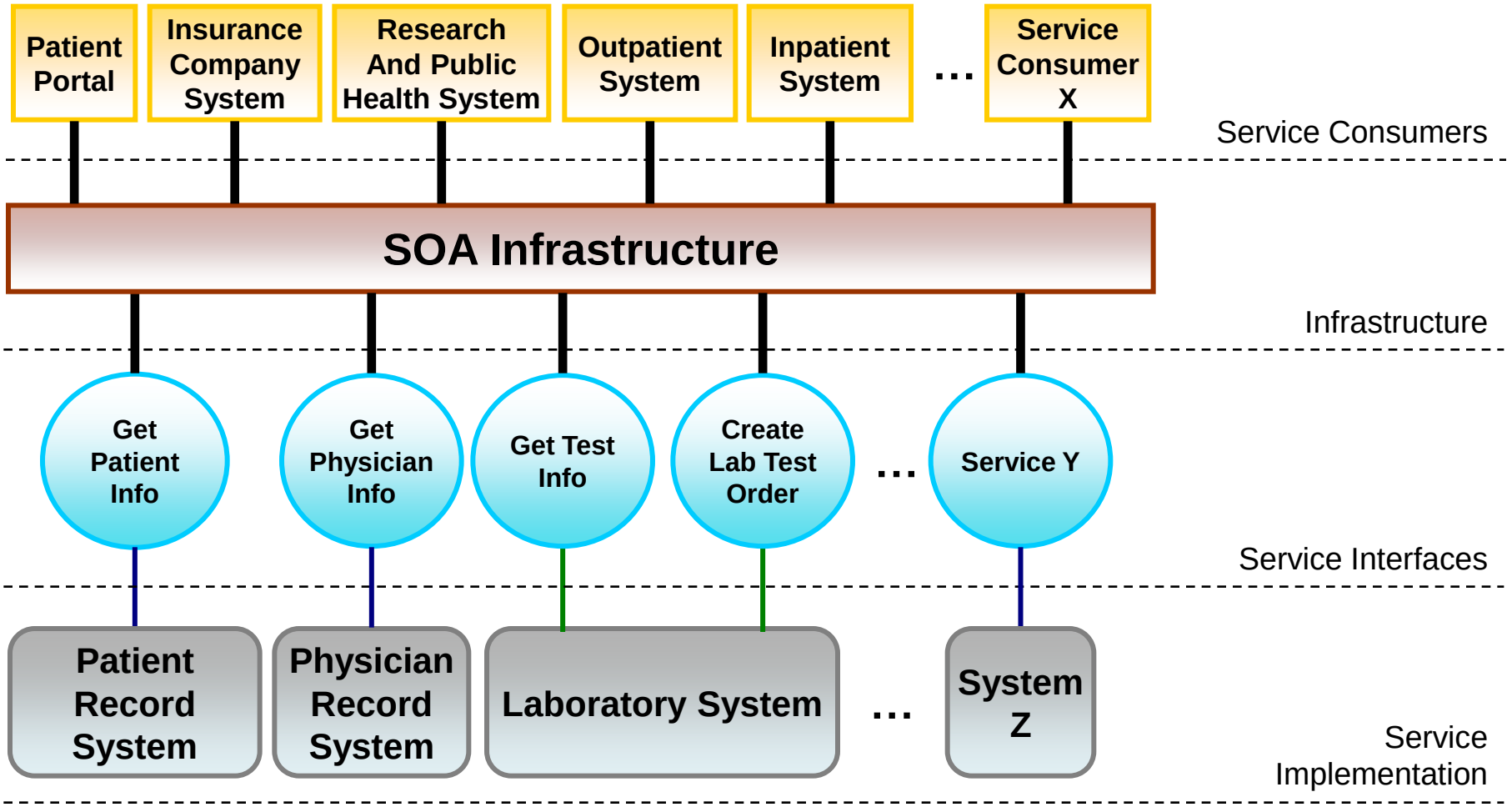
Services are reusable components that represent business tasks, e.g.

- Look up patient information
- Validate credit card
- Get test results
- Schedule appointment

Services can be

- Globally distributed across organizations
- Reconfigured into new business processes

A Notional Service-Oriented System Architecture



Intersection Between Software Product Lines and SOA

Software product lines support systematic reuse by using core assets with a production plan to enable the rapid generation of new products

SOA exposes services that can be reused by a variety of consumers, enabling:

- Agility, adaptability, cost efficiency and legacy leverage

SOA services can become core assets within a product line

Software product line framework identifies 29 required practice areas

- We initially identify decision points that need to be addressed in 4 practice areas if services are to be used as core assets

SOA services can provide significant leverage as variability mechanisms in a product line

- We initially identify 5 variability mechanisms

Agenda

Intersection of software product lines and SOA

Selected practice areas and decision points 

Variation mechanisms

Conclusions and next steps

Selected Practice Areas

Architecture

Using externally available software

Mining existing assets

Testing

Practice Area: Architecture

Specific SOA standards, interfaces and technologies

- Type of standards: web services, REST, proprietary standards
- Specific implementations: eg web services has 250 different standards

Handling of basic SOA operations

- Decisions on responsibility for operations (infrastructure, services, consumer)
  - Discovery
    - Design time
    - Use of broker; how broker is to be implemented
  - Composition and enforcement mechanisms
  - Invocation: routing, mediation, process orchestration, complex event processing

Practice Area: Using Externally Available Software

Decision Points

- How are the services to be accessed, what standards do they support, what outputs do they return
- Do the external services meet the functionality and quality of service requirements of the software product line
- What type of testing has been performed on the services, at what level
- How appropriate and effective is the service level agreement
- What types of mechanisms are built into the services to handle variations
- Do the external services have an option to establish variability points
- Can variability be handled by the infrastructure or by consumers of services

Practice Area: Mining Existing Assets

Does it make sense to migrate the legacy system to an SOA environment?

What services make sense to develop?

What legacy system components can be used to implement these service?

What changes to components are needed to accomplish the migration?

What migration strategies are most appropriate?

What are the preliminary estimates of cost and risk?

What is an ideal pilot project that can help address some of these risks?

Practice Area: Testing- 1

Service provider perspective

- Regression tests cover both conventional interfaces as well as service interfaces to make sure that changes made for one set of users do not affect the other set of users
- Functional tests consider potentially unknown users and uses of the functionality of the service.
- Greater number of consumers requires additional security testing, stress testing and load testing.
- Regression testing needs to verify whether existing service-level agreements (SLAs) are affected.
- Service providers have to maintain separate instances of their service interfaces as well as their service implementation so that test data does not affect production data.

Practice Area: Testing-2

Service consumer perspective

- service consumers need to develop and test their systems to consider the case when services are completely unavailable.
- external services
- no control over
  - changes made to the service,
  - release cycles,
  - shutdown.

Service consumers will have to be tested every time there is a new service release

Agenda

Intersection of software product lines and SOA

Selected practice areas and decision points

Variation mechanisms 

Conclusions and next steps

Parameters Invoke Service

Parameters invoke different variations of a service, such as treatment responsibilities that may depend on role (physician, nurse, technician)

- a single service is invoked
- the parameters sent to the service are modified for the appropriate behavior

In health care domain, the service “OrderTest” can have variability to enable carrying out different laboratory tests.

- Variability mechanisms can allow for differential input and output
  - Input: list of test names/codes along with the notification rules
  - Output: list of orderIDs corresponding to the specific test ordered

Infrastructure Services Hide Variability

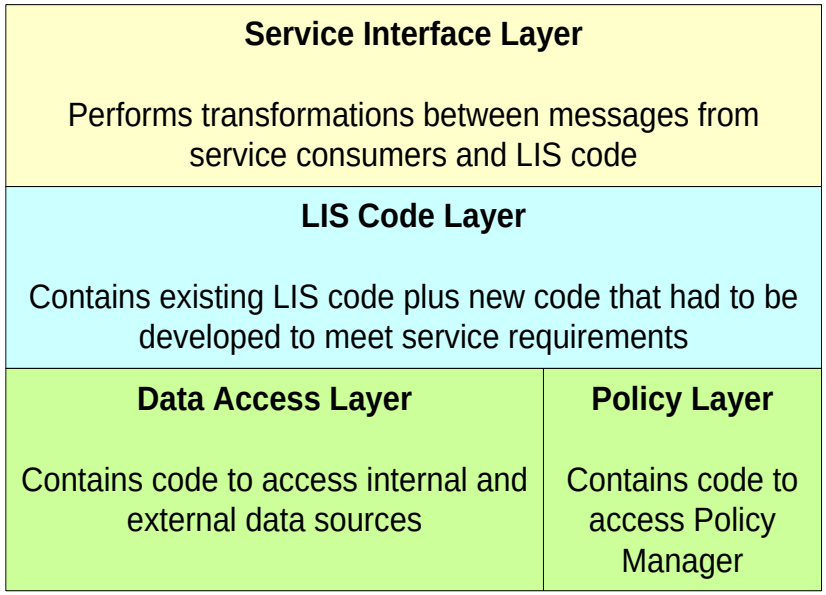
Common tasks become services that are delegated to the infrastructure.

- Examples include role based identity management and data formatting.
  - In the health domain, different sets of actors will require very different access and authorization privileges.
    - Health care insurers can access financial information
    - physicians can access detailed treatment information
    - research organizations will only be able to see information that is completely anonymous.
  - -the infrastructure can determine authorization privileges for different roles

Encapsulating Variability Within a Service

This approach isolates core service functionality from aspects that are either highly changeable, or in the case of an SPL, potential variation points.

Service layers can be created for the interface, core service code, data access and in this case, access to a policy manager infrastructure service



Differential Composition of Atomic Services

Services are often developed as atomic services that perform specific tasks. In software product lines, different configurations are required.

Composing services from a number of atomic services enables variability

- composition of web services can be delegated to the infrastructure through
  - a standard such as WS-BPEL (Web Services Business process Execution Language)
  - proprietary or custom developed Business Process Management (BPM) functionality

Differential composition enables applications or products in a product line to be developed through the integration of functionality from existing services

Different Protocols for Interface Implementations

The same business functionality can be implemented through different interfaces.

Example: ordering laboratory tests

- internal consumers can use an internal EJB implementation
- external consumers requiring the internet may need to use more standard web service implementations with greater firewall protection.

Agenda

Intersection of software product lines and SOA

Selected practice areas and decision points

Variation mechanisms

Conclusions and next steps

Conclusions and Next Steps

Examine rest of software product line practice areas for decision points and engineering tradeoffs

Expand potential set of variation mechanisms

Perform proof of concept analyses of the relevance of specific technologies, tools and methods to the context for which they were developed can also be instrumental in building up a body of knowledge in the area

- Sidharth Surana of Carnegie Mellon has completed a proof of concept analysis of the relevance of different variation mechanisms for a simple product line example

Validation and updating of the initial mappings, more complete mapping of services to SPL product line practices, and empirical research on actual SOA based SPL implementations.

NO WARRANTY

THIS CARNEGIE MELLON UNIVERSITY AND SOFTWARE ENGINEERING INSTITUTE MATERIAL IS FURNISHED ON AN "AS-IS" BASIS. CARNEGIE MELLON UNIVERSITY MAKES NO WARRANTIES OF ANY KIND, EITHER EXPRESSED OR IMPLIED, AS TO ANY MATTER INCLUDING, BUT NOT LIMITED TO, WARRANTY OF FITNESS FOR PURPOSE OR MERCHANTABILITY, EXCLUSIVITY, OR RESULTS OBTAINED FROM USE OF THE MATERIAL. CARNEGIE MELLON UNIVERSITY DOES NOT MAKE ANY WARRANTY OF ANY KIND WITH RESPECT TO FREEDOM FROM PATENT, TRADEMARK, OR COPYRIGHT INFRINGEMENT.

Use of any trademarks in this presentation is not intended in any way to infringe on the rights of the trademark holder.

This Presentation may be reproduced in its entirety, without modification, and freely distributed in written or electronic form without requesting formal permission. Permission is required for any other use. Requests for permission should be directed to the Software Engineering Institute at permission@sei.cmu.edu.

This work was created in the performance of Federal Government Contract Number FA8721-05-C-0003 with Carnegie Mellon University for the operation of the Software Engineering Institute, a federally funded research and development center. The Government of the United States has a royalty-free government-purpose license to use, duplicate, or disclose the work, in whole or in part and in any manner, and to have or permit others to do so, for government purposes pursuant to the copyright license under the clause at 252.227-7013.