THE ARCHITECTURE JOURNAL

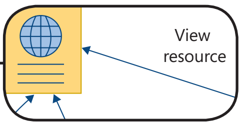
Journal 5

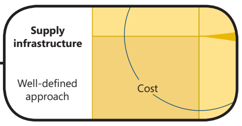
Integration Interchange

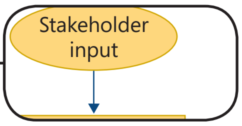
Microsoft®

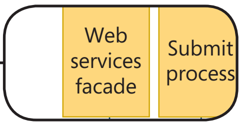
Contents

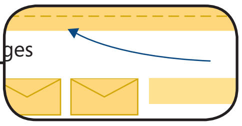
THE ARCHITECTURE JOURNAL
Journal 5

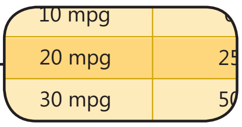











10 mpg
20 mpg
25
30 mpg
50

Editor's Note

Dear Architect,

Over the last 18 months I've been fortunate to witness the evolution of the *Architecture Journal* from its inception to this current edition: Journal 5. It has grown in many ways, especially as a vehicle for architects across the globe to share ideas, learning, and unique perspectives. One key indicator that this vehicle is getting a life of its own is the sense of continuity that is emerging across the articles—references and commentary on articles in previous editions and commitments by authors to write sequels to the topics they introduce in this edition. It encouraged me to go spelunking for old editions and raised my curiosity for future ones.

Architecture in the systems world is as vast as the galaxy, and architects who operate in it have unique challenges like no other role that comes to mind easily. What makes it even more challenging is that only a handful of educational institutions offer a formal degree program in this discipline. Unlike their counterparts in the building world with framed diplomas hanging on the walls, architectural knowledge is not so easy to acquire. It's a refined discipline in which only the brave Jedi knights with special aptitude have chosen to take on.

The rapid rate of change in the business world and speed of technology innovation dictates that architects need a medium to stay current, exchange ideas with peers, and grow. Only individuals comfortable working with ambiguity, with knowledge, and experience in many disciplines, and who enjoy juggling the requirements of various stakeholders, tend to venture into this exciting and exigent frontier. The *Journal* is a key medium that goes a long way to make this possible.

While reading the six articles and admiring their graphics in this edition, I learned new things and found perspective on perplexing questions that sometimes keep me awake at night. The article by Richard Veryard and Philip Boxer made we wander through the streets of an agile metropolis seeking parallels with SOA governance space. I stepped through a comprehensive preflight checklist from Anna Liu and Ian Gorton for evaluating technologies for my services-based integration needs. If I had to solve picking the right transportation alternatives for Web services, I would take the route of "Planes, Trains, and Automobiles" by Simon Guest. For a moment there, I thought I was reading about metropolitan transportation planning.

Every article in this edition is rich and engaging and sets the rhythm for the "Think Ahead, Learn More, Solve Now" theme of the newly launched Architect Resource Center at Microsoft.com (www.microsoft.com/architecture).

I'm confident this edition will make you think, learn something new, and help you grow like it did for me. I hope it also motivates you to use the *Journal* as a medium to share your knowledge and perspective.
Enjoy!

Gurpreet S. Pall

Foreword

Dear Architect,

It was clear by the second issue of *Journal*, now called *The Architecture Journal*, that demand was high for this kind of Microsoft publication. As editor, I received much encouragement from friends, colleagues, and customers to make this a lasting initiative instead of a one-shot wonder. My aim has always been to develop a strong community of like-minded individuals around this journal. To be accepted in the marketplace it had to provide a reputable platform for authors to express their thoughts and views in an open-minded way with as little interference from Microsoft as possible, while being credible and financially viable because of Microsoft's strong backing.

I'm glad to say that the *Journal* will still be with us for a long while. In the time since Journal 4 was published and the release of this latest issue, Journal 5, the journal has transitioned out of its incubation phase to become a key part of Microsoft's architect community development strategy. The *Journal* features prominently on Microsoft's Architect Resource Center (visit www.microsoft.com/architecture), and we now provide free subscriptions to the printed version. We're investing in the professionalism that the *Journal* requires by creating a team dedicated to producing and publishing it in collaboration with Fawcette Technical Publications. These developments are focused on making sure you, our readers, and our authors get outstanding value and experiences from reading the *Journal*.

The *Best of Journal* interim issue successfully transitioned us from the original "large" format to this regular magazine format. Tens of thousands of copies have been distributed this year at TechEd conferences in the U.S. and Europe! At the same time we're seeing a lot of momentum and interest in architecture, so much so that specialist architecture talks at these conferences compete on a par with traditional developer talks. It's an exciting time to be in this space, and you can count on the *Journal* to lead the way!

Finally, you'll notice that I'm writing this foreword and not the editorial as expected. Well, that's because Gurpreet Pall did such a good job with his version of the foreword, we decided to switch.

As always, if you're interested in writing for this publication, please send me a brief outline of your topic and your resume to asehmi@microsoft.com.

Wishing you good reading!

Arvindra Sehmi

An Introduction to Topic Maps

by Kal Ahmed and Graham Moore

Summary

The ISO international standard topic maps paradigm describes a way in which complex relationships between abstract concepts and real-world resources can be described and interchanged using a standard XML syntax. Here we'll introduce the topic maps paradigm in the context of the ISO standard, present the principal components of the topic map model, and demonstrate how the standard processing components of scope and topic merging give additional power to this model.

Topic maps were developed originally in the late 1990s as a way to represent back-of-the-book index structures so that multiple indexes from different sources could be merged. However, the developers quickly realized that with a little additional generalization, they could create a metamodel with potentially far wider application. The result of that work was published in 1999 as ISO/IEC 13250 – Topic Navigation Maps.

In addition to describing the basic model of topic maps and the requirements for a topic map processor, the first edition of ISO 13250 included an interchange syntax based on SGML and the hypermedia linking language known as HyTime. The second edition, published in 2002 (see Resources), added an interchange syntax based on XML and XLink, and to date this syntax has the widest support in topic map processing products. We'll describe the syntax here.

Today there are a number of implementations of the standard, both open source and proprietary, for a number of languages and platforms including the .NET platform. A *topic map* consists of a collection of topics, each of which represents some concept. Topics are related to each other by associations that are typed *n*-ary combinations of topics. A topic may also be related to any number of resources by its occurrences.

Figure 1 shows the three fundamentals of topic maps. You can see how the distinction between topic-to-topic and topic-to-resource relationships enables a partitioning of the model into a topic space that contains only topics and associations between topics, and a resource space that contains the resources related to topics. This partitioning is interesting because it allows a topic map developed for one set of resources to be repurposed to index a different set of resources, and thus the topic map can be considered to be a portable form of knowledge.

Share and Share Alike

Unlike domain-specific models, the topic map model has no predefined set of types. Instead, individual topic map authors or groups of authors in

Figure 1 The three fundamentals of topic maps: topics, associations, and occurrences

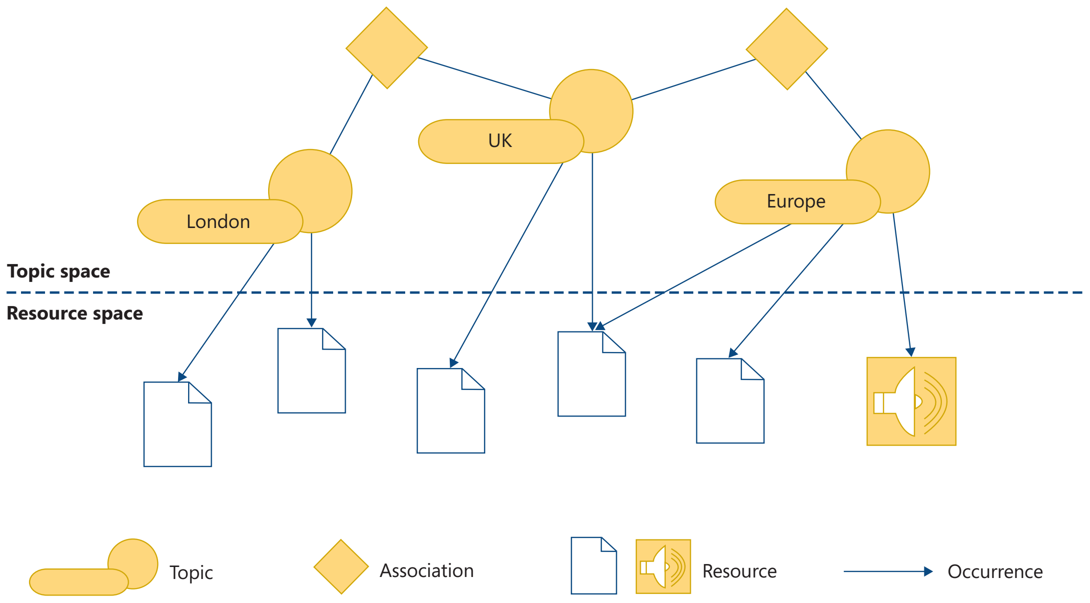
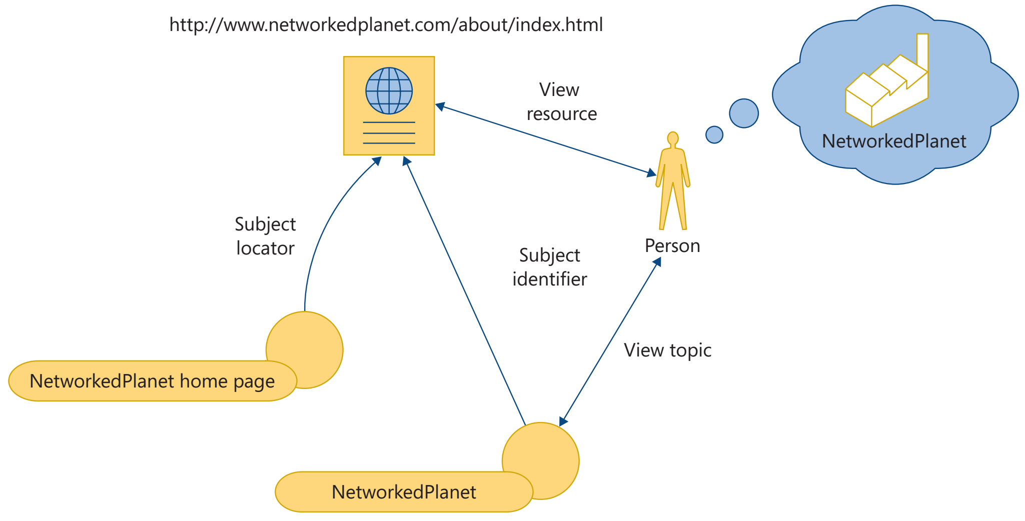


Figure 2 Subject locator and subject identifier



a community or practice can define the model for their domain of interest and share those models with other authors from other domains.

We believe that for many end users, a good topic maps application will conceal much if not all of the topic maps mechanism, allowing users to instead concentrate on the domain model(s) that they work with. However, the topic maps model and the topic maps standard do provide a number of benefits that can be surfaced in applications and that can be unique selling points.

The core topic maps metaphor of topics, occurrences, and associations strikes a balance between being compact and easy to understand and providing enough basic infrastructure to allow users to translate their mental model of a domain into a topic map model. Other forms of data and information organization, such as RDF and the relational model, may have a simpler model still, but then require the user to create infrastructure for common procedures, such as labeling an item with some names, defining a class structure, or creating *n*-ary relationships between items.

As described previously, the topic maps model has a clear distinction between the domain model, expressed as topics and associations between topics, and the indexed resources, expressed as occurrences that link topics to resources. Three major benefits can be derived from this structure:

- The topic map can act as a high-level overview of the domain knowledge contained in a set of resources. In this way the topic map can serve not only as a guide to locating resources for the expert, but also as a way for experts to model their knowledge in a structured way that allows nonexperts to grasp the basic concepts and their relationships before diving down into the resources that provide more detail.
- A topic map can be partitioned easily depending on the resources to be made available. Some publishers use a topic map-based index of large sets of resources and then dynamically create the appropriate index when they publish a subset of those resources. With some thoughtful modeling it is even possible to create different layers of detail in a topic map and to differentiate between information products based on the indexing and navigation features that they provide as well as the informational content of the products.
- Topic maps that index different resource sets can be combined easily. This feature can be used to allow organizations to *import* third-party data and indexes and seamlessly integrate their own data and indexes.

As already noted, the topic maps standard does not come with a predefined ontology. (Note that the word *ontology* in this context means the system of types of topics, occurrences, and associations that together define the classes of things and relationships between things that are documented by a topic map.) There is no restriction on the domains to which topic maps can be applied and relatively few constraints even on the modeling approach taken.

Easy Interchange

We have seen topic maps used to model temporal relationships between events, relationships between abstract concepts and their depictions, and forms of first-order logic, as well as more traditional relationships such as thesauri, and controlled vocabularies and business information.

For many users, the fact that topic maps can be interchanged using a standard XML-based syntax provides a strong benefit in improving the portability of their data between applications and platforms. In addition, the XML interchange syntax allows easy integration of topic map information exchange within the Web services architecture.

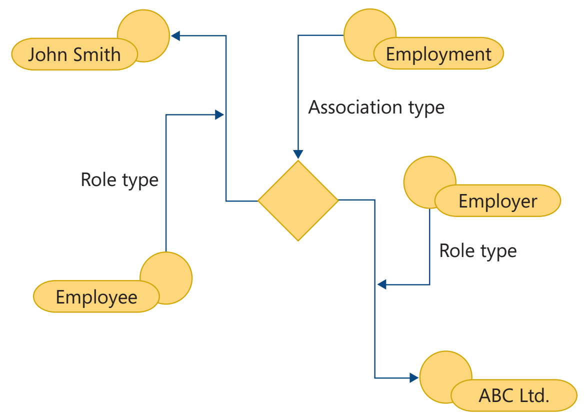
There are three principal benefits that system architects and developers can gain from the application of the topic map paradigm, and they can be summed up as “flexibility, flexibility, and flexibility.”

Topic maps provide the metamodel on which a completely flexible application model can be built. Creating new types of business objects can be achieved by adding data to the ontology that constructs the topic map. Because the ontology is itself expressed as topics and associations between topics, extension of the ontology becomes an issue of adding data, not an issue of redesigning the underlying storage schema. This issue makes it possible to modify the data model used by an application without the need to upgrade deployed persistent stores.

With the application data stored in a standardized and extensible metamodel, the path is open to enable much simpler third-party application integration and extension. This model would allow a third-party developer to define its own data and code extensions to an application without relying on the core application’s schema supporting their extension-specific data structures.

In addition to allowing third-party extensions to the application schema, the flexibility of the topic map structure can be used to allow users to create their own extensions. This flexibility has two effects: it

Figure 3 The structure of an association



enables applications to be highly customizable, and it enables development of horizontal applications that can be integrated more easily into existing environments.

For an example of the customizable effect, the flexibility allows much easier integration of customer-specific data systems. We see this effect as being the key to vertical applications that can be deployed more easily for multiple customers. For example, one successful topic map application was produced by a publisher of legal information for the financial services market. The unique selling point of their topic maps-based product is that they can integrate their customers' marketing and procedures documentation with the legal information they provide.

A *topic* is a machine-processable representation of a concept. The topic maps standard does not restrict the set of concepts that can be represented as topics in any way. Typically, topics are used to represent electronic resources (such as documents, Web pages, and Web services) and nonelectronic resources (such as people or places). Topics can be used equally to represent things that have no tangible form at all, such as companies, events, and abstract concepts like "pensions" or "insurance."

Forms of Identity

Topics have four principal forms of identity. A topic can have zero or more of each of these forms of identity and thus can be identified within a topic map system by a number of different ways.

Identity as a topic resource in a serialized topic map. When a topic map is represented in a serialized form for interchange, each topic is assigned a Uniform Resource Identifier (URI) that is unique across that topic map. These URIs are used principally for deserializing references between topics. Such identifiers are referred to as source locators.

Identity as a human-readable label. A topic can have any number of topic names. Names act as labels for human consumption and can be either text or a reference to some nontextual representation (for example, an icon, a sound clip, an animation, and so on). The scope mechanism (described later) allows for the case of homonyms (where a single word is used to refer to two or more different concepts).

Identity by reference. When a topic is used to represent a resource that already has its own unique URI, that URI can be used as part of the identity of the topic, which is simply a way of telling the processing agent that "this topic stands for that resource." In the topic map standard, this form of identifier is known as a subject locator.

Identity by description. Topics can be used to represent a concept that does not have its own unique URI. Many of the things that a topic can represent could never have a unique URI because they are not things that a computer can resolve a reference to. For example, a person may have any number of database records about them or online biographies or pictures, but none of those addressable resources is the person; they are merely some form of descriptor for the person. In the topic map standard, this form of identifier is known as a subject identifier, and the resource that the subject identifier resolves to is known as a subject indicator. Topic maps allow the use of URI references to such descriptive resources as a form of identity. Obviously, it is important that the topic map author chooses unambiguous descriptive resources for this purpose, and we'll return to this issue later.

The distinction between the latter two of these forms of identity can be confusing. Consider the URL www.networkedplanet.com/about/index.htm. This URL links to a Web page that describes the company NetworkedPlanet. This URL could be used as the subject identifier for a topic named "The company NetworkedPlanet" because it resolves to a resource that describes the concept of the company. However, if we want to talk about the concept "The About page on the Web site, www.networkedplanet.com," we want a topic whose subject really is the resource at the address: www.networkedplanet.com/about/index.html, and so we would then use the same URI as a subject locator (see Figure 2).

Types and Names

The key difference between a subject identifier and a subject locator is that a subject identifier requires human interpretation of a resource to determine the concept that a topic represents; whereas, a subject locator simply points to the concept that the topic represents (see Figure 2). The arrow on the left shows the use of a resource as a subject locator. The arrow in the center shows the use of the same resource as a subject identifier. The arrows to the right show the role of the human being in the interpretation of a subject identifier.

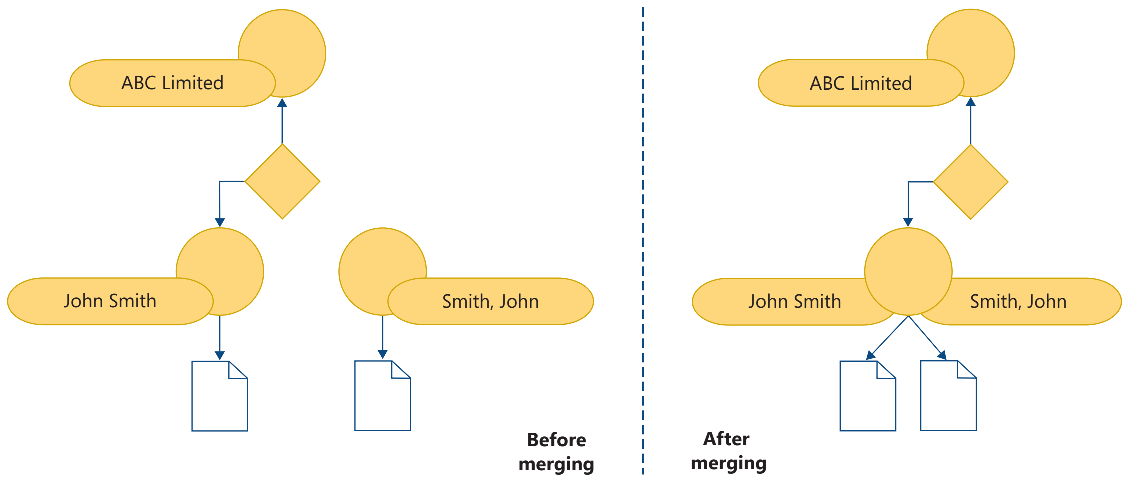
Although a single topic can have many forms of identity, it is important to note that each separate identifier can resolve to only one topic. The merging rules of topic maps (described later) enforces this one-to-many relationship between topics and their identifiers.

In addition to these forms of identity, a topic can also have any number of types and any number of names. The types of a topic define the class (or classes) of concept that the concept represented by the topic belongs to. Types are treated in topic maps as concepts in their own right, hence, every type is represented by a topic. The type of a topic is specified simply by a privileged form of relationship between the topic that represents the instance and the topic that represents the type.

The names of a topic define a set of labels for a topic. Every name has a hierarchical structure. At the root is the base name, which has a string representation. It is the base name string value that is used to determine topic identity by label. A base name is also a container for any number of alternate forms (known as variant names). The alternate forms of a name may be either string values or references to resources, allowing representations such as icons or sound clips to be referenced as variant names. Base names and variant names can be given a context (or scope) in which they are valid, allowing a topic map-aware application to select the best name for presentation to a user in a given situation. We will cover scope later.

Associations are the general form for the representation of relationships between topics in a topic map. An association can be thought of as an *n*-ary aggregate of topics. That is, an association is a grouping of topics with no implied direction or order; and there is no restriction on the number of topics that can be grouped together.

Figure 4 The topic merging process



An association can be assigned a type (again defined by a topic) that specifies the nature of the relationship represented by the association. In addition, each topic that participates in the association plays a typed role that specifies the way in which the topic participates.

For example, to describe the relationship between a person, John Smith, and the company he works for, ABC Limited, we would create an association typed by the topic Employment and with role types Employee—for the role played by John Smith—and Employer—for the role played by ABC Limited (see Figure 3).

Occurrences and Scope

Occurrences are used to represent or refer to information about a concept represented by a topic. Occurrences can be used either to store string data within the topic map, or to reference any kind of Web-addressable resource external to the topic map. No restriction is placed on what type of resource is addressed by an occurrence. It may be a static HTML page, an HTML page generated by ASP, a Web service, or any other type of resource. Neither are occurrences restricted to the HTTP protocol—any address encoded as a URI can be used to address an external resource.

Once again, occurrences can be typed, using a topic to express the occurrence type, and a scope of validity can also be assigned to an occurrence. Like names, an association can be assigned a scope in which it is valid, which may be used by a topic map-aware application to determine whether or not to display the information represented by the association to a user in a given situation.

In the topic map standard *scope* refers to a constraint or a context in which something is said about a topic. The way in which such statements about topics are made are by adding a name to the topic, specifying an occurrence for a topic, or by creating an association between topics (in which case the statement applies to all of the topics in the association).

In many cases statements are not always true, but are dependent on a context. For example, we make statements such as "ABC Limited was top vendor of widgets in Q2 2004" or "Fred says that ABC Limited is a good investment." In the first statement the context is a temporal context, and in the second case the context is a quotation context. More prosaically, context is often used to facilitate multilingual interfaces so the concept dog may have the label *dog* in the context of the English language, *le chien* in French, and *das Hund* in German.

In a topic map, scope is defined by a collection of topics that can be assigned to a name, an occurrence, or an association. The default scope (where no set is assigned) is known as the unconstrained scope and simply means that the name, occurrence, or association is always valid.

When a topic map-aware application encounters a name, occurrence, or association that has a scope assigned to it, the application should make use of information it has about the current operating context and compare that information against the scope information contained in the topic map to determine if the construct is valid and whether or not it should be presented to the user.

In the current edition of ISO 13250, the mechanics for processing scope against an application context are not constrained by the standard, and for many topic map developers this lack of constraint is seen as a shortcoming as it can make it more difficult to exchange topic maps that use scope. The next revision to the standard will recommend that a scope that consists of multiple topics should be processed such that the scoped construct is valid only if the application determines that all of the topics in the scope apply to the current application context.

Topic Merging

Automatic topic merging is a key feature of topic maps that brings many benefits to topic map development and to applications that make use of topic maps for managing and exchanging data.

The principle behind topic merging is that in any given topic map, each subject described by the topic map must be represented by one and only one topic in the topic map. This principle means that it is the responsibility of the topic map processor to attempt to identify the situation in which two topics represent the same subject and to process them so that only one topic remains. This is the process of *merging*.

Identifying when two topics represent the same subject is achieved by applying heuristics. The topic maps standard defines a set of basic heuristics:

1. If two topics share the same source locator, then they have been parsed from the same topic map source and must be considered to represent the same concept.
2. If two topics have the same subject locator, then they both identify the same network resource as being the thing that they represent.

Listing 1. The XML syntax defines a topicMap element that contains any number of topic and association elements; in this case there is syntax for membership, group, singer, and guitarist.

```
-->
<!-- The Clash is a Band -->
<topic id="clash">
<instanceOf>
  <topicRef xlink:href="#band"/>
</instanceOf>
<baseName>
  <baseNameString>The Clash</baseNameString>
</baseName>
</topic>
<!-- Joe Strummer is a Person (note multiple names)
-->
<topic id="joe-strummer">
<instanceOf>
  <topicRef xlink:href="#person"/>
</instanceOf>
<baseName>
  <scope>
    <topicRef xlink:href="stage-name"/>
  </scope>

  <baseNameString>Joe Strummer</baseNameString>
</baseName>
<baseName>
  <baseNameString>Joseph Mellor</baseNameString>
</baseName>
</topic>
<!—
Joe Strummer is a member of The Clash -->
  Note separate member elements
  used for the different roles
  played
-->

<association>
<instanceOf>
  <topicRef xlink:href="#membership"/>
</instanceOf>
<member>
<roleSpec>
  <topicRef xlink:href="#group"/>
</roleSpec>
<topicRef xlink:href="#clash"/>
</member>
<member>
<roleSpec>
  <topicRef xlink:href="#singer"/>
</roleSpec>
<topicRef xlink:href="#joe-strummer"/>
</member>
<member>
<roleSpec>
  <topicRef xlink:href="#guitarist"/>
</roleSpec>
  <topicRef xlink:href="#joe-strummer"/>
</member>
</association>
</topicMap>
```

3. If two topics have the same subject indicator, then they are both using the same resource to describe the concept that they represent and must be considered to represent the same concept.
4. If two topics each have a base name with the same string representation and the scope of the base names are the same set of topics, then the topics must be considered to represent the same concept.
5. Finally, a topic map application may make use of any domain-specific information it has, to determine that two topics represent the same concept.

Item 3 in this list raises the importance of selecting a good resource as the description for a concept. If the description is somehow ambiguous or if the resource addressed is not well defined enough, it is possible that two different topic map authors might use the same resource as a descriptor for different concepts, leading to undesired merging. In our experience, good resources for subject descriptors are those created specifically to describe a single subject—for example, the pages at wikipedia.org, or pages created by the topic map author(s), or by a community of practitioners to define a controlled vocabulary.

Item 4 has proven to be controversial in the topic map community as it relies on what many consider to be a relatively weak form of identity: the name for a concept in some language. The mapping of words to concepts in a language is a complex affair, and one has challenges in multiple words having different meanings (homonyms), not to mention localization challenges! In the next version of the ISO standard, the restrictions on name-based identity will be tightened still further to require an author to explicitly flag a topic name as being one that should be used to confer an identity (the default being that a name shall not confer identity to its topic).

Item 5 allows for applications to extend the topic maps standard's set of merging criteria with application-specific criteria. These could include criteria based on more than a straightforward string or URI comparison. For example, an application might know that "The Duke" and "John Wayne" are names for the same actor and merge two topics on that basis. Having identified the topics to be merged, the merging process defines the process of replacing those two (or more) topics with a single topic. The single topic that results from the merging process has all of the identifiers, names (including variant names), and occurrences of the topics that are merged. In addition, the result topic replaces the merged topics wherever they are referenced (that is, in any associations, scopes, or types that they appear in). This process is shown schematically in Figure 4.

The merging to two (or more) topic maps is simply the process of combining their sets of topics and associations and then applying the topic merging rules to the result.

The XTM Syntax

As noted previously, the ISO topic maps standard defines two standard interchange syntaxes: one SGML based and the other XML based. The XML syntax defines a topicMap element that contains any number of topic and association elements. A simple example of a topic map in XML Topic Map (XTM) syntax looks like this:

```
<topicMap xmlns=
  http://www.topicmaps.org/
  xtm/1.0/ xmlns:xlink=
  "http://www.w3.org/1999/
  xlink">
  <topic id="band">
  <baseName>
   <baseNameString>Band
   </baseNameString>
  </baseName>
  </topic>
  <topic id="person">
  <baseName>
   <baseNameString>Person
   </baseNameString>
  </baseName>
  </topic>
<!--
```

And Listing 1 shows you similar syntax for membership, group, singer, and guitarist.

We will not go into the details of the syntax here. (see Resources for a link to the original XML topic maps specification produced by TopicMaps.org—which was subsequently adopted by ISO).

Note that the XTM syntax does not impose the merging restrictions that are required of a topic map processor. This allows XTM to be created easily, but requires that any processor that reads an XTM file must detect topics that must be merged and apply merging rules as the XTM file is parsed. When an XTM file is known to be fully merged (that is, it doesn't contain topic elements representing topics that should be merged), the topic map model it contains can be accessed easily using standard XML processing tools such as XSLT and XQuery. However, it is not the case that standard XML processing tools can be easily applied to XTM files where merging is required.

Despite the issues with merging, the XTM syntax serves the basic need of allowing interchange between conformant topic map processing applications. In addition, the syntax and merging rules together are sufficiently flexible to even allow parts of a topic map to be serialized as separate XTM documents and later recombined through merging (see Resources).

As we have hopefully demonstrated up to this point the topic maps standard provides a very flexible base architecture for a wide variety of information and knowledge management applications. This flexibility can lead to confusion and constant reinvention of basic modeling approaches. To address this issue, we advocate the development and use of patterns within topic map applications. We divide patterns into two broad categories: topic map design patterns that are patterns for modeling topic map data and topic map application patterns, which are architectural patterns for the use of topic map processing systems.

The basic concept of a topic map design pattern borrows heavily from design patterns in software engineering. A topic map design pattern provides a focused and reusable ontology that addresses a single issue. There are a couple of interesting differences, however.

A topic map design pattern can be more prescriptive than a software design pattern, as it should specify the subject identifier URIs for the key topics used by the pattern. In this way every implementation of a particu-

Hierarchical Classification Pattern

Some basic patterns for hierarchical and facetted classification are supported by several topic map processing applications. The Hierarchical Classification Pattern uses a modeling property of topic maps in which every topic, association, and occurrence type is itself a topic.

Problem statement. Many classification systems consist of one or more hierarchies of subjects. A number of different hierarchical relationships are possible: part-whole, broader-narrower, and so on. Although the relationships may be different, the hierarchical semantics remain the same. An application working with multiple hierarchical relationship types requires a way to identify all of these types.

Pattern description. The pattern given here for modeling a hierarchical classification system uses one topic to represent each class in the system. The hierarchy is then modeled by creating associations between subordinate and superordinate classes. However, it is recognized that there are a wide variety of different hierarchical relationships. For this reason, the type of the associations between the subordinate and superordinate classes are not defined by this pattern. Instead, this pattern defines the type of all such types, and the type of all role types for both subordinate and superordinate role players.

The other issue is how to relate the items classified by this scheme (the instances) to the topics that represent the classes. If an instance is represented by a topic, then an association should be made between the topic representing the class and the topic representing the instance. For this purpose, topic types are introduced to represent the classification of an instance ("Classified as") and the roles played in the relationship ("Classification" and "Instance"). If the instance is not represented as a topic, then an occurrence should be used, in which case the "Instance" type can be used as an occurrence type rather than as an association role type.

Analysis. This pattern creates a means of flagging an association type as being a hierarchical relationship and of indicating which role is the superordinate and which is the subordinate role. This analysis may be done externally to the topic map, which defines the association and role types, enabling a preexisting topic map to be integrated without the need to edit it.

The classification semantics of the "Classified as," "Classification," and "Instance" types can be omitted from this pattern, where classification is not the purpose of the hierarchy. For this reason, those subjects are defined in a separate set of Published Subject Identifiers (PSIs)—with a different base URI—from the hierarchy-defining subjects.

PSIs for the Hierarchical Classification Pattern

These PSIs are used by the Hierarchical Classification Pattern:

- *Hierarchical relation type* (www.techquila.com/psi/hierarchy/#hierarchical-relation-type/) – A type of association type. Associations that are typed by a topic that is an instance of this type represent a parent-child relationship between two or more topics.
- *Superordinate role type* (www.techquila.com/psi/hierarchy/#superordinate-role-type/) – A type of association role type. The player(s) of a role that is typed by an instance of this type in an association of the type Hierarchical Relation Type is the upper element in the hierarchy.
- *Subordinate role type* (www.techquila.com/psi/hierarchy/#subordinate-role-type) – An association role type. The player(s) of a role that is typed by an instance of this type in an association of the type Hierarchical Relation Type is the subordinate element in the hierarchy.
- *Classified as* (www.techquila.com/psi/classification/#classifed-as/) – An association type asserting the relationship between a topic representing a class in a classification system (playing the role Classification) and one or more topics representing instances of that class (playing the role Instances).
- *Classification* (www.techquila.com/psi/classification/#classification/) – The role played by a topic representing a class in a classification scheme.
- *Instance* (www.techquila.com/psi/classification/#instance) – The role played by a topic representing a subject that is classified under a classification scheme. •

lar pattern in a topic map can be recognized instantly by the presence of topics with the URIs specified by the pattern.

As a topic map is purely data, behaviors related to a topic map design pattern are implemented not in the topic map itself but in the processing software that makes use of the topic map data. Some design patterns may prescribe a particular set of behaviors for processing applications; others may describe only the data model and leave open the way in which the application processes the data model.

Topic Trees

Some basic patterns taken from Library Science have been defined by one of the authors and are supported by a number of topic map processing applications (see Resources). These patterns include patterns for hierarchical and facetted classification. (See the sidebar, "Hierarchical Classification Pattern" for an example of one such pattern).

The Hierarchical Classification Pattern makes use of a very useful modeling property of topic maps, which is that every topic, association, and occurrence type is itself a topic. This feature allows the ontology of a topic map application to be annotated using the same structure as is used to populate the ontology itself, and can be used to great effect in design patterns by allowing an existing topic map ontology to be annotated using the pattern "metaontology" rather than modified.

This pattern enables an application to process a set of associations between topics as representing a hierarchy. For example, it may display the topics arranged into a tree view.

Topic map application patterns provide high-level architectural patterns and principally concentrate on the integration of a topic map processing system with other data systems and applications. These patterns include patterns for representing information from external data systems as topic map data, patterns for the import of information from external data systems, and patterns for the export and display of topic map data. (The details of topic map applications are beyond the scope of this article).

At the time of this writing more work is being done within ISO both on the topic maps standard itself and on a suite of companion standards. Although ISO/IEC 13250 has been through a revision, the core of the standard has remained unchanged since 1999—exhibiting a fair degree of stability in comparison to many Internet standards! However, the ISO committee has decided that the next version of the standard will be a significant overhaul in the way the standard is presented and a minor overhaul of the standard itself (see Resources for a link to the Hierarchical Classification Pattern).

The ISO/IEC 13250 standard is to be divided into a number of separate parts: a non-normative introduction, a formal description of the underlying data model of topic maps, an XML/XLink-based interchange syntax with a description of the process of deserializing the syntax into an instance of the data model and serializing the data model into a document conforming to the interchange syntax, and a canonicalization algorithm for the data model that can be used in topic map processor conformance testing. It is hoped that this organization will make the standard more reader friendly and will add features that were missing originally and were felt to be important for future developments (specifically, the formal model specification and the canonicalization algorithm).

Changes to the standard include the ability to apply datatypes to occurrence values, including the ability to embed XML; the ability to declare a subset of the names of a topic as names to be used for determining topic identity; a clearer model of scope; and a definition of the interchange syntax in W3C XML Schema and Relax-NG as well as XML DTD.

What's Ahead

In addition to the changes to ISO/IEC 13250, the committee has also commenced work on two companion standards. ISO/IEC 18408, Topic Maps Query Language (TMQL), will define a language for querying the topic map data model, allowing the selection of both topic map constructs (such as topics and associations) and of the data carried by them (for example, topic name or occurrence values).

ISO/IEC 19756, Topic Maps Constraint Language (TMCL), defines a schema language for topic maps that would allow the schema author to constrain the constructs that can appear in a topic map and how they must relate to one another. As with XML a schema language for topic maps enables both validation and also smarter, schema-driven editing applications.

Both of these standards are currently in an early stage of development with requirements defined, and, in the case of TMQL, an initial proposal for the language has been created. Work on the core standard and on the query and constraint languages can be followed on the ISO Topic Maps Web site (see Resources). •

Resources

SO/IEC
www.isotopicmaps.org
Home of SC34/WG3 Information Association

www.y12.doe.gov/sgml/sc34/document/ 0322_files/iso13250-2nd-ed-v2.pdf "ISO/IEC 13250, Topic Maps (Second Edition)," M. Biezunski, S.Newcomb, and S. Pepper (ed.) (May 2002)

Techquila
www.techquila.com/topicmapster.html
"TMShare — Topic Map Fragment Exchange in a Peer-To-Peer Application," Kal Ahmed

http://techquila.com/tmsinia.htm
"Topic Map Design Patterns for Information Architecture," Kal Ahmed

Topic Maps 4 Java (TM4J)
www.tm4j.org
Welcome to the TM4J Project

XTM TopicMaps.org
www.topicmaps.org/xtm/1.0/, "XML Topic Maps (XTM) 1.0," Steve Pepper and Graham Moore, eds. (TopicMaps.org 2001)

About the Authors

Kal Ahmed is cofounder of Networked Planet Limited (www.networkedplanet.com), a developer of topic map tools and topic maps-based applications for the .NET platform. He has worked in SGML and XML information management for 10 years, in both software development and consultancy, and on the open source Java topic map toolkit, TM4J as well as contributed to development of the ISO standard.

Graham Moore is cofounder of Networked Planet Limited, and he has worked for eight years in the areas of information, content, and knowledge management as a developer, researcher, and consultant. He has been CTO of STEP, vice president research and development at empolis GmbH, and chief scientist at Ontopia AS.

Metropolis and SOA Governance

by Richard Veryard and Philip Boxer

Summary

From time to space, the defining technology of the past thousand years was the clock, and the prevailing technological imperative has been to save time to make things go faster or better than before. Approaches to building and governing large, complex service-oriented systems is not unlike designing and managing the complexities of a large city.

Modern mechanical clocks with falling weights were invented around a thousand years ago. The invention is often attributed to Pope Sylvester II, who was an accomplished mathematician and scientist before becoming pope. Under the Rule of St. Benedict, mediaeval monasteries used the clock to regulate labor and prayer. Lewis Mumford traces the origins of the Industrial Revolution to the Rule of St. Benedict and to the domination of the clock. Charlie Chaplin's film, *Modern Times*, shows (in exaggerated form) the relentless power of the clock over the production-line worker. Business process engineering in the late twentieth century was focused on reducing cycle time and eliminating waiting. The key slogan: Just in time.

Of course, it is too early to say what the defining technology of the next thousand years will turn out to be, but there are already some signs of a shift from an emphasis on time toward an emphasis on difference. The Internet, for example, creates new kinds of difference in the relationships between people and organizations. Business organizations operate as differentiated nodes or clusters within complex networked ecosystems; social and institutional cohesion is mapped against the coordinates of complex abstract dimensions of difference.

Our understanding of complexity itself rests on the recognition that once we go beyond a certain threshold of difference in the behaviors of systems it becomes impossible to predict their composite behavior. Thus, in the service economy we expect service-oriented systems to emerge that are increasingly large and increasingly complex, but that are also capable of behaviors that are increasingly differentiated, which as we shall see, is one of the key challenges of service-oriented architecture (SOA).

The city or metropolis is another large complex system that many of us encounter in our everyday lives and that we typically experience as differentiated in its behavior. Our familiarity with cities makes the metropolis a good starting place for working out the best approach to building and governing such large complex systems. Furthermore, many of the key issues for the design and governance of large complex service-oriented systems arise also in the field of urban design, where they have been debated for decades (although without reaching consensus).

As it happens, the latest heavyweight contribution to the debate on physical architecture and urban design comes from Christopher Alexander, whose long-awaited work on the nature of order has finally been published. Alexander has had a profound influence on software engineering for many years—his early work on the synthesis of form influenced the structured methods of Yourdon and deMarco, while his middle work on patterns was taken up enthusiastically by large numbers of software engineers, especially in the OO world.

From Cities to SOA

In the twentieth century, two of the best writers on the nature of cities were Lewis Mumford and Jane Jacobs. Mumford thought a well-ordered city needed central planning and infrastructure, while Jacobs took a more anarchist position (see Table 1). Here are some of the issues raised in the city debate (see Resources).

- *Adaptability* – In nineteenth-century England, Manchester was highly adapted to the cotton industry, but failed to adapt to later waves of industrialization. Meanwhile, Birmingham was far more

Table 1 SOA governance questions from Pat Helland

(In contrast to metropolitan city governance...) IT governance is not so mature.	**Enterprises might learn a lot by looking at how cities manage the difficult process of resource allocation.**
Who makes the tough choices in IT? Is it the CEO, the CIO, the business unit leaders, techies, or perhaps committees?	What proposals are projected to pay for themselves?
Are priorities established based on cost, flexibility, or asset utilization?	What is the timeframe and risk analysis around these projections?
What is success and how is it measured?	What in your organization is sacred?
Are we seeking cost reductions, business process transparency, or competitive advantage?	What resources remain after funding these efforts?
	What balance of short-term, long-term, and speculative investments are right within the specific corporate culture?
	These problems are common for metropolitan and IT environments.

adaptable, and this adaptability enabled it to accommodate a series of industrial innovations.

- *Complexity* – A city contains a vast quantity of social and commercial interaction. A living city allows for many different levels of such interaction and for meaningful clusters and subclusters to emerge, which form an abstract hierarchy.
- *Governance* – City planning requires orchestration of developments large and small, balancing local initiative and autonomy against global coherence.

There are some strong parallels between town planning and SOA, which make it reasonable to translate ideas and experience from urban design into the SOA domain:

- *The distribution of design* – Detailed design decisions are taken within different organizations, each following its own agenda (for example, commercial or political goals).
- *The constancy of change* – Elements of the whole are being redesigned and reconfigured constantly, and new elements are being added constantly. Structures must evolve in robust ways.
- *The need for progressive improvement* – Each design increment should not only make local improvements, but should have a positive effect on the whole.
- *The recursive nature of the architecture* – Similar design tasks must be carried out at different scales (levels of granularity).

In a recent article in the *Microsoft Architects Journal* (see Resources online at www.ftponline.com), Pat Helland offers an extended analysis comparing the planning and management of IT with the planning and management of cities. He argues that IT governance has a lot to learn from city governance (see Table 1) and raises some interesting parallels between urban design and SOA (see Table 2).

Helland's article makes this argument: Progress requires standardization (according to Helland, people didn't even wash properly until they had standard clothing); standardization is associated with commoditization; standardization requires concentration of power (and if this involves pathological distortions of socioeconomic relations, so be it); infrastructure requires central investment (since we may regard infrastructure as an act of local standardization, it follows that it must involve concentration of power); and central investment preserves the "sacred." Let's look at the steps of Helland's argument in detail.

Standardization. Progress should involve enriching people's lives, while many acts of standardization impoverish them instead. Not everyone is willing to regard Wal-Mart as the epitome of progress. Living systems are heterogeneous. Helland's utopia appears to be a relatively homogeneous one. Citizens even all smell the same. The exclusion of antisocial odors is engineered though standard interchangeable clothing (although doubtless soap and clean water contribute something as well).

Modern production methods allow for mass customization, which involves a separation of production into two layers: a homogeneous layer of mass production and a heterogeneous layer of customization. In urban design, the standardized stuff is what goes underneath the pavement—standard utilities such as water and power, for example. The human stuff goes above the pavement. It is a question for city governance to decide what should or may go under the pavement.

The Nature of Order: Christopher Alexander's Manifesto

The *Nature of Order* (see Resources online at www.ftponline.com)—in its complete four volumes—has implications for the involvement of people in the design of buildings and in the detailed ways in which this involvement is likely to be successful or unsuccessful. It has implications for the flow of money. It has implications for the handling of architectural detail and for the successful integration of structural engineering into the framework of design. ... It affects virtually every part of the profession we now know as architecture, and it indicates necessity for change, in almost all of them. There is no question, that under the impact of this theory, architecture will be deeply changed, and it will be changed for the better. •

A Question of Architecture

The articulation of a complex system into two layers (one homogeneous, one hetero) is an architectural question. Obviously there are companies with a significant commercial stake in this question. It is therefore worth being suspicious of an articulation that is presented as given by some historical tradition or technical imperative. (For a practical example, consider phone companies that would like to regard the location of mobile phone masts, or towers, as mere infrastructure, to be decided on technical grounds and requiring no public consultation. However, people are becoming concerned about the radiation from these masts, especially near homes and schools, and this politicizes their location.)

Apparently pure technocratic architectural judgments often conceal a commercial agenda. One of the functions of governance is to maintain a "level playing field" between different commercial agendas. Therefore, regulators often aspire to intervene at the architectural level.

It is conceivable that something could look homogeneous from the supply side and heterogeneous from the demand side, or vice versa, so the boundary is itself relative to some supply/demand formulation. Technology is constantly creating homogeneous/heterogeneous splits; for example, Voice over IP (VOIP) technology creates a split between devices that care whether a bitstream represents voice or data (and therefore regard the traffic as heterogeneous) and devices that don't care (and therefore regard the traffic as homogeneous).

One plausible basis for articulation of layers is the differential rate of change. It may appear to make sense to standardize and regulate the slow-moving layer and allow greater diversity in the fast-moving layer. However, an ecological perspective informs us that the slow moving dominates the fast moving. This perspective entails a new role for architecture: To maintain appropriate stratification of layers and coupling between elements within and across layers, and to operate at a higher level of abstraction, implementing evidence-based design policies.

Existing approaches to defining architectures may not work very well even for the homogenous bits. They certainly don't work for the heterogeneous bits, and they also don't help with defining the boundary between the homogeneous layer(s) and the heterogeneous layer(s). A consequence of our argument is that the boundary between homogeneous and heterogeneous (as illustrated previously) is a proper focus of attention for architecture. We shall come back to what this means in practice.

Commoditization. We cannot avoid the commoditization of our lives, but we should be wary of its dangers. (A possible reconciliation of commoditization with human values is provided by Albert Borgmann—see Resources.) Fortunately, we no longer have to put up with one-size-fits-all software. Situated software—software designed in and for a particular social situation or context—resists the traditional software engineering pressure toward generalization and apparently disregards the economics of scale/scope. Instead it works solely within a collaborative sociotechnical system (the "community"); the conditions for the success of the software (including meaning and trust) are co-created by the members of the community.

One of the earliest forms of situated software was the spreadsheet. Power users built themselves complicated structures using Visicalc or Lotus 123 or Excel. These were essentially nontransferable artifacts with many hidden assumptions, but they served a useful purpose within a given context, which illustrates the fact that situated software is assisted by the existence of tools and platforms that provide generalized support for situated software.

Supporting Rich Diversity

The overwhelming success of the spreadsheet was because it performed a useful function, while leaving the user free to create context-specific meanings. However, the spreadsheet was also limited in that these meanings were private and undocumented, and attempts to turn spreadsheets into shared artifacts typically failed.

This is where the software factory comes in. There is a great opportunity here to produce software tools capable of supporting a rich diversity of end-user demand. A Domain-Specific Language (DSL) can be a way of bridging and holding open the gap between the general public and the context-specific or private and maintain the dynamic interaction between supply and demand. This dynamic needs to be driven by the way the end user defines the relationship between domains and their business as a whole.

Table 2 Key parallels between cities and IT from Pat Helland

Cities	IT shops
Factories or buildings	Applications
Transportation	Communication
Manufactured goods	Structured data
Manufactured assemblies	Virtual enterprises
Retail and distribution	Business process
Urban infrastructure	IT infrastructure
City government	IT governance

The service economy is a complex ecosystem. Service-oriented solutions are essentially systems of systems, and their composition should be mindful of complex systems theory. To maintain requisite variety (and therefore survival of the fittest) in such an ecosystem, we need diversity at all levels of abstraction.

Concentration of Power. The Wal-Mart economic system is unsustainable. It destroys the fabric of small shops on which a rich urban life depends. Cities are somewhat paradoxical. On the one hand, a city is itself already a concentration of human activity, but the concentration processes are unstable and can result in highly dysfunctional urban forms.

Historically, cities have had walls to keep out unwanted visitors. Elsewhere, Helland has advocated the fortress model of computing. However, here he seems to be envisioning a continuous metropolitan fabric, where one city fades into the next (as Manchester merges into Salford).

Central Investment. Helland's plea for central investment (the Mumford position) provides a justification for corporate central planning and investment in IT. Many large and small organizations attempt to impose central IT planning. However, in many organizations this is a losing battle. The situation of the IT industry emerges from millions of small procurement decisions, and is closer to the idea of anarchic procurement (the Jacobs position).

Nikos Salingaros draws on Alexander to describe how the Mumford-Jacobs dispute can be resolved, but by adopting an approach that goes beyond simply attempting to reconcile the top-down with the bottom-up.

A modern version of the sacred may be found in Borgmann's notion of focal things and practices. The role of urban/system governance would be to create/preserve a space in which these focal things and practices can be developed and honored.

In an agile demand economy, the source of the sacred is demand, which contrasts with a supply-side logic based on a presumption of symmetric demand in which markets are defined to reflect the supplier so that formations of demand are symmetrical to the formations of supply.

In talking about virtual enterprises, Helland writes: "you have to consider the context in which the part will be used. Is weight or ruggedness the principle concern?" Helland argues that standards are the key to enabling component providers to leverage the cost of optimization across a broader market, and this leverage can be understood as supply-side logic. However, this issue goes beyond standardization (here extended to business component models, enabling them to be encapsulated as component capabilities and orchestrated as a part of larger assemblies of processes) and opens up the granularity of component capabilities with respect to each other. That is, what is the repertoire of alternative component capabilities available? This concept needs to be understood from the demand-side as well.

Table 3 Implications of asymmetry: the three dilemmas

Symmetry	Assumptions	Implications of Asymmetry
Technology=Product	The first symmetrical assumption is that the first three challenges are all lined up. Thus, these three challenges collapse into a single dimension defined by the technology.	With SOA, we are increasingly confronted with businesses that are nothing more than a technology platform for other businesses (from Microsoft itself downward). The simple alignment doesn't work. Instead, they are pushed into some form of stratification.
Business=Solution	The business rules and procedures espoused by the supplier match the ways the services are to be used.	Rail maintenance is supposed to provide reliable and safe railways. In the UK, Network Rail (formerly Railtrack) takes input services from engineering companies and turns them into output services for train-operating companies. It has proved to be extremely difficult to align the input requirements against the output requirements.
Customer demand= Customer experience	The banking fantasy of straight-through processing is based on symmetry and shared values all along the value chain.	Contrast this characteristic with the situation in the pharmaceutical industry, where a drug company's set of relationships with GPs and pharmacists are of quite a different nature to those of the GPs and pharmacists with their patients, despite the tendency of the drug companies to assume that it should be otherwise.

The Driving Force

In considering business process, this argument is extended by analogy to the need for standardization and interchangeability in data and operations: "people cheerfully accept standard stuff and customization is rare and expensive. But business process is still largely hand-crafted. There are poor standards..." And so the argument goes for standardization providing a basis of extending a supply-side logic deeper into the provision of services, with business process becoming the driving force that dictates the shape and form of applications "as surely as Wal-Mart drives the standards for many, many manufactured goods."

This argument doesn't meet the challenge of the aforementioned retail cycle (see Resources). This cycle describes the emergence of a new form of supply-demand relationship (destination), which expands to become a new form of offering alongside others (comparison) before it becomes commoditized (cost) and ultimately embedded into the user's context (custom). From here, the ground is prepared for a new cycle (the transitional cusp) and so on. This cycle is a dynamic process in which the supply side is constantly learning new forms of supply in response to a demand that is always evolving—and never fully satisfied. Asymmetric demand describes the demand in its particular context of use, and this "something always left to be desired" is a structural deficit that is always there driving the development of markets. Commoditization is only the supply-side part of the story; the real issue is the way the dynamics of the formation of demand itself has to be supported.

"THE CITY OR METROPOLIS IS ANOTHER LARGE COMPLEX SYSTEM THAT MANY OF US ENCOUNTER IN OUR EVERYDAY LIVES AND THAT WE TYPICALLY EXPERIENCE AS DIFFERENTIATED IN ITS BEHAVIOR"

For several decades, Christopher Alexander has been exploring alternatives to conventional architectural practice. His latest work, the *Nature of Order*, was published last year (see the sidebar, "The Nature of Order: Christopher Alexander's Manifesto").

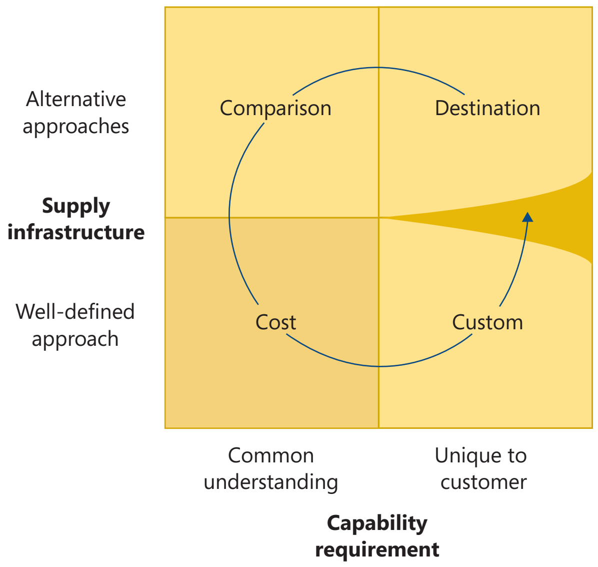
According to Alexander, large complex systems cannot be produced by a conventional design process, either top down or bottom up (see Figure 1). Instead, they emerge from an extended and collaborative (evolutionary) process. Order and coherence comes from the rules that govern this evolutionary process, which can be broken down into discrete steps that may either preserve structure and wholeness or destroy it. Structure and wholeness is articulated as a recursive system of centers.

Services can easily be considered as centers of value (see Resources online at www.ftponline.com). Services can be composed into composite services, with service orchestration (hopefully) yielding coherence between recursive levels.

A service-oriented enterprise can then be understood as a continuous corporate web of services. The architectural properties of this enterprise depend on the numerous collaborating processes that bring about its composition. If these are appropriately structure-preserving, then the enterprise can become both increasingly differentiated and increasingly integrated, without loss of coherence.

Alexander is highly critical of the conventional governance over city planning and urban design, and highly critical of the inflexible and inhuman results of directed composition. His ideas on design and order are, we believe, consistent with the needs of collaborative composition, as outlined here.

Figure 1 Capability requirement



Structural Implications

A business or value chain is composed in a geometric structure. In SOA, we design a business or value chain as a network of services, which is a powerful geometrical pattern. However, there may be many possible network geometries capable of satisfying a given business requirement, all of which count as satisfying the principles of SOA—for example, hub/spoke or peer to peer.

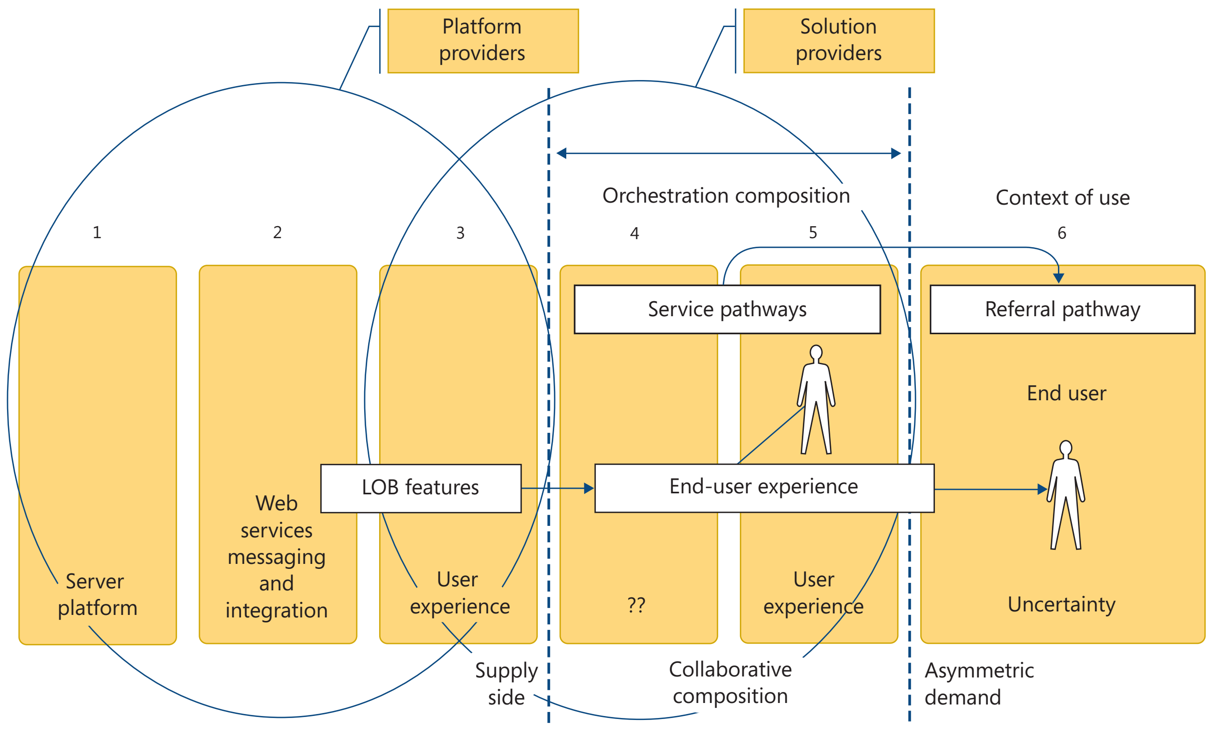
A key characteristic of SOA is stratification. A business process is composed of services from a set of lower-level services presented as a platform. A good example of a business platform is the set of retail services offered by Amazon and eBay. Other service providers have built further retail/logistical services on top of the Amazon/eBay platforms.

Each platform is in turn built on even lower services. At the lower levels there may be collections of IT-based services, known as the enterprise service bus (ESB). There may also be sociotechnical service platforms, such as call centers. Some of these lower layers of the stack may appear to be purely technical services; however, a more complete picture should reveal the existence of an IT organization maintaining the platform. In other words, it too is a sociotechnical platform that includes its administrators and programmers.

Thus, we have a stratified geometry in which a person tackling a problem at a given level is presented with a collection of available services, formed into a virtual platform. You can think of this collection as a business stack, with one platform stacked on top of another. And while the SOA principles may provide some geometrical guidance, and mandate certain geometrical patterns, there is still a design job to determine the form of geometry most appropriate to supporting the demanded service. This design job may be easy when the requirement is trivial, but gets harder as the complexity increases.

In many situations, the demand side has more variation than a human designer (or design team) can accommodate. (We characterize

Figure 2 Collaborative composition in the insurance value chain—based on Microsoft IVC (Source: CBDI Forum, November 2004)



this situation as an asymmetry of demand, which calls for a process of asymmetric design.) Under these circumstances, we need to go still further and start thinking about variable geometry solutions, where the geometry itself can be adapted on demand.

For example, in the past we have assumed that granularity has to be fixed at design time, but we can conceive of a Web service platform that detects patterns of demand-side composition; defines new component services automatically, describing and publishing these new services in real time; and notifies likely users of the new service, complete with an incentive to switch to new ways of orchestrating them in support of demand-side composition. We can conceive of such a Web service platform analyzing the message content of a certain service and producing a substitute service with a smaller footprint that would satisfy most of the uses in a more elegant way.

We use the term value landscape to refer to the distribution of cost, benefit, and know-how across a complex market ecosystem, such as the insurance industry, for a given level of risk. Technology (including SOA) influences business geometry because it not only affects transaction costs, but also the way know-how can be leveraged in relation to demand. The shape of the value landscape changes (has already started to change) as the result of B2B, B2C, P2P, and BPO. Companies that once occupied safe market positions may find their commercial advantage slipping away, or they may find themselves cut off from their former customers or supply chains.

Taking Aim

Let's suppose an insurance company has these strategic aims:

- *Profitability, short-term viability.* To deliver the maximum service value as cost effectively as possible, using available input services and technologies as efficiently as possible, with minimum costs/risks of change
- *Adaptability, medium-term viability.* To understand and respond to changing demand for insurance services, and to trends in cost and risk, both internally and across the industry; to develop and deploy new services to exploit new business opportunities and avoid emerging business threats
- *Survival, long-term viability.* Making sure the core business proposition remains valid and doesn't get eroded by more agile players; taking strategic action in relation to structural changes in the insurance industry

If we are doing business geometry for an insurance company, we need to think about the insurance industry as an evolving ecosystem. We need an as-is model of the present ecosystem (largely based on pre-SOA technologies) and a to-be model of an emerging ecosystem (based on the effects of SOA). We can expect the pre-SOA ecosystem to evolve into some form of post-SOA ecosystem; although, we may not have much idea which of the possible changes is going to happen first. To satisfy all three strategic aims, an insurance company needs to exploit the pre-SOA ecosystem and prepare for the post-SOA ecosystem.

Note that this situation may force the insurance company to implement a variable geometry across its business stack, both in the organizational platform and in the underlying IT platform. Otherwise, it will either have to operate suboptimally for an extended period, or incur significant organization costs and IT costs every time the industry takes another step toward SOA. Variable geometry involves a dynamic collaboration (collaborative composition) between an efficient supply side and a variable (asymmetric) demand side (see Table 2).

Asymmetry means that the forms of demand are increasingly specific to the context in which they arise. The first asymmetry involves separating out technology from the supply of specific products (see

Figure 2). This separation requires modeling of possible behaviors that can be supported (so Microsoft or car manufacturing has to modularize itself in support of families of technology use).

The second asymmetry requires separating business models that can organize supply from the solutions that are on offer. This separation requires modeling of the possible forms of business geometry (so rail maintenance or retail services have to use a franchise model to allow the variation in business organization to accommodate the variety of ways in which the service needs to be implemented).

And the third asymmetry requires separating the different contexts of use. This separation requires modeling the possible forms of demand (so that financial or care services are having to take up the way the through-time wealth/conditions are managed in a way that responds to different forms of context of use).

Taking the Lead

These asymmetries are summarized in Figure 3, and it is worth considering what happens if they are ignored. In the first asymmetry the product is defined by the technology, which is typical of the early stages in the emergence of new technologies. (Do you remember how we used to have to use mobile phones?). In the second asymmetry the solution for the customer is defined by the way the business is organized. (Do you remember how large businesses used to relate to their customers before CRM?). And in the third asymmetry the solution to the problem presented by the customer is assumed to be what the customer needs. (Have you ever received a prescription from the doctor that turns out only to treat the symptom?). We see the major competitive impact of SOA being that it changes what the supplier can afford to ignore from the customer's perspective.

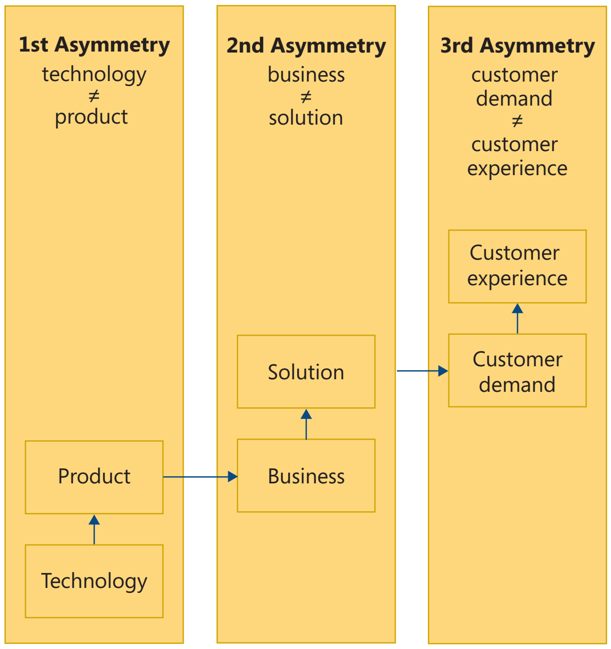
To become better at capturing asymmetric forms of demand, an organization needs leadership that will enable it to do two things: take power to the edge of the organization and develop an agile infrastructure. Taking power to the edge of the organization means the people at the edge of the organization with the relationship to the asymmetric demand must be able to organize the business model they need to capture that demand. Developing an agile infrastructure means providing business services that can be orchestrated and composed at the edge in response to the particular forms of demand they are targeting. This leadership then allows the supply side of a business to extract economies of scale or scope when providing support across multiple business models.

"IT IS CONCEIVABLE THAT SOMETHING COULD LOOK HOMOGENEOUS FROM THE SUPPLY SIDE AND HETEROGENEOUS FROM THE DEMAND SIDE, OR VICE VERSA"

Let's look at a health care example. Asymmetry in the variety of conditions that people have, and the ways in which they reveal themselves in people's lives over time, is ever increasing. Meanwhile hospitals and clinics are having to become ever more efficient in how they administer particular care pathways.

We see this challenge particularly in relation to conditions that are chronic. Thus, patients don't produce conditions that fit the treatments that are available, while organizing acute care systems around the treatment of chronic conditions becomes exorbitantly expensive. Patients' conditions are relentlessly asymmetric from the point of view of the medical specialisms trying to care for them. For example, acute surgery can often be traced to an earlier failure to provide timely prophylactic treatment.

Figure 3 Three asymmetries of demand



Getting power to the edge in health care means the funding following the patient and the doctors having the ability to craft a treatment plan that is particular to the patient's condition. This situation creates a double challenge for providing health care. Not only must it increase the flexibility with which its component services may be made available to patients, but also the doctors have to have a much greater involvement in formulating whole-life strategies of providing health care that can be tailor made to a patient's condition over time, and for the delivery of which they can be held accountable. Where this can be done, the total cost of care is reduced.

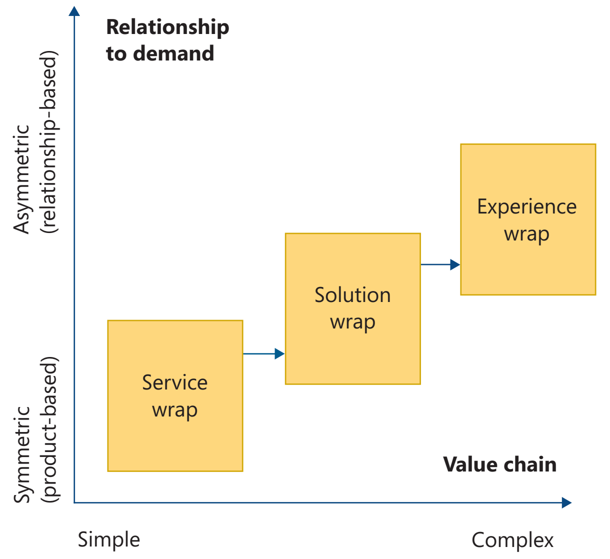
Addressing Asymmetry

A health care purchasing and supplies agency (PASA) was responsible for the supplies of equipment to a clinical service. They were concerned about the side effects of minimizing the costs of these supplies—reduced investment in the industry and a vicious circle of decline in the quality of the clinical service itself. They decided that the conventional symmetric design wasn't working for them; in an attempt to improve the quality of the clinical service itself, they decided to consider developing an approach that addressed the asymmetric nature of the clinic's demands. They conducted an initial pilot study to establish the feasibility of an asymmetric design process.

What PASA accepted was that they needed to address the demand side of the clinic and establish how best to satisfy its needs. Within this context of use, they could then address the question of the costs of supply.

A national process was set up by the modernization agency. This process ran six pathfinder projects, each of which aimed to establish how to effect change in each context. It was the national project's task to secure long-term support and funding for the change process in the light of the learning and results of the pathfinders. This process ultimately involved modeling the regional and national impact of the changes on NHS and social services budgets.

Figure 4 The supply-demand road map



From the point of view of the supply to the clinics, the demand-side modeling was of the referral pathways and the services offered by each clinic in response to the demands arising from those referral pathways. The supply-side modeling was of the organization of the clinic itself, together with its use of suppliers, to establish how the one was aligned to the other. Where this cut came between the supply side and the demand side was a function of who the client was, and what was their change agenda. Nevertheless, in examining the referral pathways and the particular ways in which they themselves had been *colonized* by suppliers, further questions were raised about the organization of primary care itself. These questions were left, however, to be taken up by a different client system at a later time—and which would have to address the interests of the strategic health authorities.

The key challenge was to give the clinicians design control over how the clinics operated (that is, power at the edge). Fundamental to this challenge was to grasp on the supply side the chronic multiepisode nature of the conditions being treated by the clinic, and on the demand side the processes of delegation and/or teaming of clinical responsibility for patients' conditions. The clinicians lacked the means of defining the different characteristics of the former and managing the complexities of the latter. Furthermore, without the means of doing these things, there was no practical way of holding the clinicians accountable for the clinical performance of their clinic. The solution was to build a reporting platform that could support doing these tasks.

This modeling involved defining the referral pathways and their characteristics, from which emerged the requirement to change the way the clinic was relating to demand, as well as defining the service propositions and clinical business models needed. In the former case, this meant establishing episode protocols for different conditions, and in the latter, changing the workflow processes between the clinic and front and back office processes in support. No realignment of the supply infrastructures was attempted, considerable gains being available simply through the way the alignment of suppliers to the demand was managed. The reporting platform built provided the means of achieving this goal. The platform allowed the clinic to define its own treatment protocols in relation to its own definitions of referred conditions. The back end of the platform was able to lift data out of the NHS environment on patients, appointments, and so on.

Supply-Demand Road Map

Figure 4 shows a typical road map for the service-based business as it responds to the competitive impact of SOA. The business tackles each form of asymmetry in turn. It uses a service wrap to decouple the product from the technology, which includes defining a different object model for the demand side, separating raw data from what we might call "cooked" data; it uses a solution wrap to decouple the solution from the business, which includes defining different rules for the demand side, separating the business logic from the orchestration of different solutions; and it uses an experience wrap to decouple the ongoing customer experience from the particular solutions bought by the customer at any one time, which includes new forms of process modeling to understand the customer experience of solutions within their particular contexts of use.

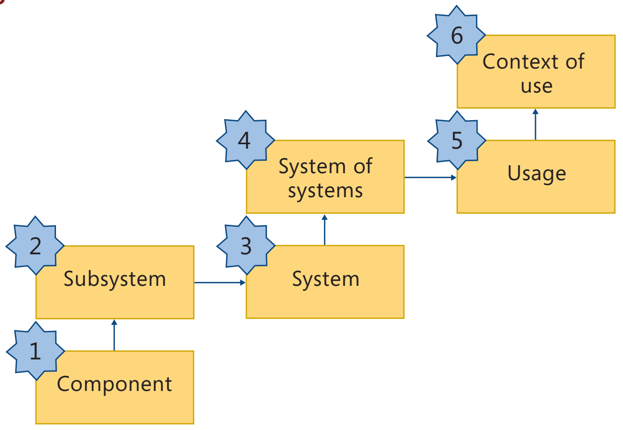
"THE CHALLENGE FOR SOFTWARE ARCHITECTS IS TO REMAIN RELEVANT TO AN SOA WORLD—TO A WORLD OF DISTRIBUTED PRODUCTION OF DISTRIBUTED SERVICES—BY PAYING ATTENTION TO THE REAL STRUCTURAL ISSUES EMERGING IN THIS ON-DEMAND WORLD"

The point about this progression is that it confronts the supplying business with the need to manage increasing complexity (and concurrency) in the way the value chain relates to the customer, hence, the competitive significance of SOA (see Table 3).

With the first symmetry, the business needs only one model because demand can be inferred from supply (or so it seems at the time!). When this symmetry is broken two models are needed: one to manage the technology and the other to manage the business. How many times has an investor of venture capital had to teach this to a start-up business?

If the full potential of a business proposition is to be realized, the second symmetry of coupling between two modules may have a significant impact on the structural cohesion and flexibility of a large system, but this coupling may be determined by fairly low-level development choices that are not even visible to the software architects (see Figure 5).

Figure 5 The stratification of models



In many organizations, software architects can be outmaneuvered and marginalized by coalitions of developers and users, and as outsourcing increases, the position of the software architect may become yet weaker—especially in those instances where contractual specifications focus on the functional requirements and underspecify the structural properties, and where there are inadequate mechanisms for the architects to verify the structural properties of the delivered software. (For example, hidden coupling that compromises the intended flexibility of a software artifact.)

So the challenge for software architects is to remain relevant to an SOA world—to a world of distributed production of distributed services—by paying attention to the real structural issues emerging in this on-demand world. Otherwise, they will be unable to contribute anything of value to the design and management of on-demand systems of systems. This leads to a need for forms of analysis that support an asymmetric design regimen and can enable explicit consideration to be given to implicit choices being made concerning geometry.

We have seen some vendors recognizing the supply-side issues of reconciling multiple Web services (for example, IBM Rational), while other platform vendors are creating the conditions for an explosion in the numbers of (behavioral) domains needing to be brought into relation with each other (for example, Microsoft). In both cases the service-oriented business is configured as a continuous fabric of services: "the corporate web". However, this result can never be achieved in one large ambitious project. It has to be achieved progressively through a continuous stream of small and medium projects.

Service Production

In the collaborative planning approach, order and coherence emerge from distributed activity with no central design authority. Each unit of procurement, development, or maintenance activity has to be regarded as a project, with project outputs being constituted as services. Thus, each project contributes something positive to the emerging corporate web of services. So what form of governance is needed to maintain architectural order?

SOA governance is required to ensure that each project satisfies the global demands of the corporate web and to ensure that there is a well-balanced mix of projects—different types and different scales (large, medium, and small).

Our discussion here has outlined the limitations of a supply-side approach to SOA governance; directed composition is limited in its capacity to respond to the full heterogeneity of demand. It leaves too great a value deficit in relation to demand, which is increasingly heterogeneous; asymmetric; and spatially, as well as temporally, differentiated. We need to take governance to the edge of the organization, acknowledging that we are engaged in processes of asymmetric design, and prepare to meet the twenty-first century on its own terms. In a subsequent article we shall open up the question of what taking "governance to the edge" means for the design of SOA infrastructures as well as for the relation to demand. •

Resources

"Centers: The Architecture of Services and the Phenomenon of Life," FTPOnline, Richard C. Murphy (March 2004)

The Component-Based Business: Plug and Play, Richard Veryard (Springer 2001)

The Culture of Cities, Lewis Mumford (Secker & Warburg, 1938)

The Death and Life of Great American Cities, Jane Jacobs (Vintage Books, 1961)

"The Information Architecture of Cities," *Journal of Information Science*, *Vol. 30, No. 2,* Andrew Coward and Nikos Salingaros (April 2004), pp. 107-118

"Metropolis," *Microsoft Architects Journal* (April, 2004)

The Nature of Order, 4 Volumes, Christopher Alexander (The Center for Environmental Structure, 2002-2004).

"Principles of Urban Structure," Nikos A. Salingaros (Delft University Press, 2005)

"The Service Based Business," *CBDI Journal,* Richard Veryard and David Sprott (2003) and "SOA Governance and Business Driven SOA," (2004)

Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, Jack Greenfield et al. (Wiley, 2004)

Technology and the Character of Contemporary Life: A Philosophical Inquiry, Albert Borgmann (Chicago, 1984).

"Triply-Articulated Modelling of the Anticipatory Enterprise," Philip Boxer and Bernie Cohen (BRL Working Paper, revised 2004)

About the Authors

Richard Veryard is a writer, management consultant, and technology analyst based in London, UK.

Philip Boxer is a strategy consultant based in the UK. The authors thank Bernie Cohen, Pat Helland, and Nikos Salingaros for comments on earlier drafts.

Services-Based Integration Technologies

by Anna Liu and Ian Gorton

Summary

Integration technologies are complex, highly technical, and diverse collections of products that operate typically in mission-critical business environments. A services-based approach to integration holds the key to seamless future integration and interoperability and can help with evaluating integration technologies.

Building an enterprise application integration (EAI) solution is difficult. These solutions need to integrate multiple business systems that were not intended to work together. Integrating such systems is difficult for many reasons, including the heterogeneity of the platforms and programming languages, the diversity and complexity of each individual business system, and the difficulty of understanding the requirements for the resulting integrated solution.

Software architects undertake a number of crucial tasks during the design of integrated enterprise applications. Among these tasks are helping understand the functional and quality requirements for the integrated applications; creating the initial architectural blueprint for the integrated applications; selecting suitable integration technologies that can fulfill the application requirements; and validating that the combination of the architecture and the integration technology used to build the enterprise-wide application is likely to be successful before a major implementation investment is made.

We'll describe a proven approach to assist architects with evaluating EAI technologies. In particular, we'll focus our discussion on evaluating integration technologies for implementing services-based integration.

SOA for Integration

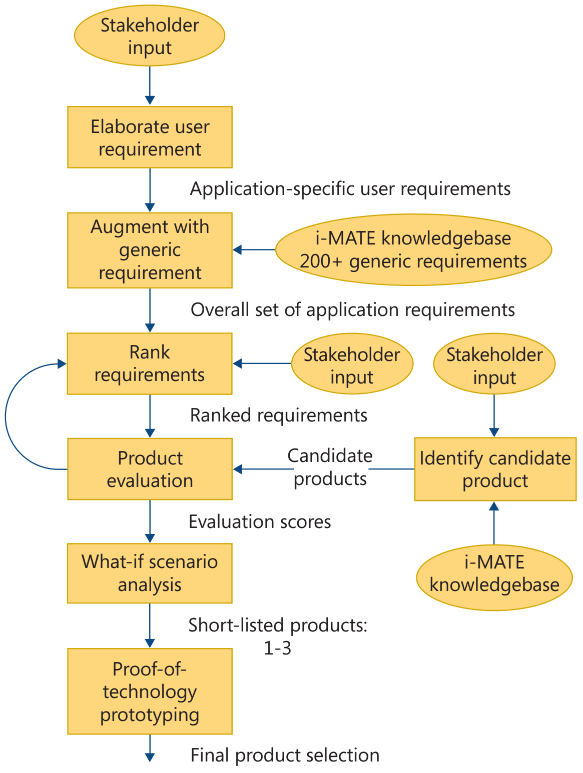
With the advent of industry standards such as Web services, service-oriented architecture (SOA) is driving a paradigm shift in many areas, including enterprise application integration. The services-based approach to integration addresses problems with integrating legacy and inflexible heterogeneous systems by enabling IT organizations to offer the functionality locked in existing applications as reusable services.

In contrast to traditional EAI, the significant characteristics of the services-based approach are well-defined, standardized *interfaces* in which consumers are provided with easily-understood and consistent access to the underlying service; *opaqueness* in which the technology and location of the application providing the functionality is hidden behind the service interface (in fact, there is no need for a fixed services provider); and *flexibility* in which both the providers of services and consumers of services can change—the service description is the only constant. As long as both the provider and consumer continue to adhere to the service description, the applications will continue to work.

Technologies for building services-based integration need to have these basic functionalities: message delivery, intelligent routing, event services, application adaptors, XML translation/data transformation, Web services support, service/process orchestration, business process management (BPM), and business activity monitoring. Further, to ensure the success of SOI, the ESB needs to have these qualities: scalability, high performance, security, and manageability.

As you can see, one of the major focuses is on using industry standards such as Web services to enable message delivery and various other advanced services, thus avoiding the problems of traditional EAI technologies, such as the use of proprietary protocols for message exchanges. In this way, services-based integration is a design pattern

Figure 1 Evaluation process


Stakeholder input
Elaborate user requirement
Application-specific user requirements
Augment with generic requirement
i-MATE knowledgebase 200+ generic requirements
Overall set of application requirements
Rank requirements
Stakeholder input
Stakeholder input
Ranked requirements
Product evaluation
Candidate products
Identify candidate product
Evaluation scores
What-if scenario analysis
i-MATE knowledgebase
Short-listed products: 1-3
Proof-of-technology prototyping
Final product selection

that ensures interoperability and true integration in any heterogeneous enterprise landscape.

There are many varied implementations of integration technologies that provide these functionalities. They range from the traditional EAI technologies with Web services features added on, to brand new implementations with inherent Web services support. Unfortunately, selecting an appropriate integration technology implementation for building SOI is not a simple proposition for most IT organizations. There are numerous reasons, but they typically revolve around these reasons:

- *Technology complexity.* Integration products are large, diverse, and literally have thousands of features and application programming interfaces. They are complex to understand, and low-level details can have serious effects on the way a product behaves. The devil is really in the detail.
- *Product differentiation.* There are tens to hundreds of products competing in the integration arena. At a superficial level, many have almost identical sets of features and capabilities. Price difference can often be very large, which further compounds the problems of selection and acquisition.
- *IT organization knowledge.* End-user organizations rarely have architects and engineers who have the necessary deep and broad understanding of integration technologies and products. It is therefore a time-consuming, expensive exercise for the organization to acquire this knowledge to choose an appropriate integration product. It also distracts key engineering staff from their mainstream, application-focused tasks.

Evaluation

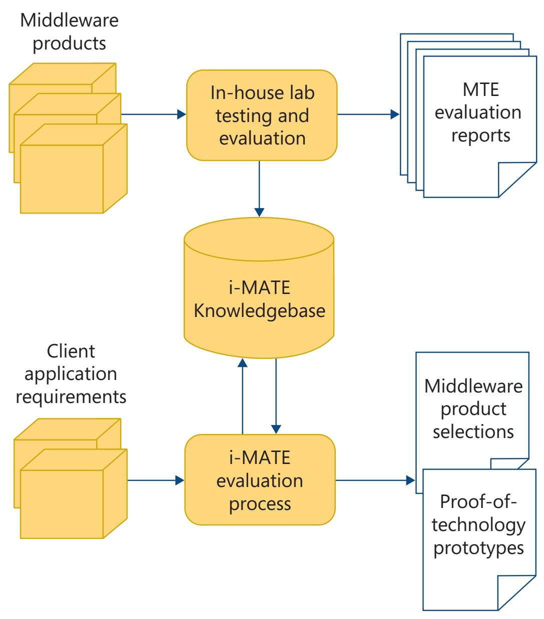
Middleware Architecture and Technology Evaluation in Internet time (i-MATE) is a specialized software engineering process for evaluating Component Off The Shelf (COTS) middleware. It is suitable for organizations operating at Level 3 in the Software Engineering Institute's Software Acquisition Capability Maturity Model, especially in its support for the User Requirements and Acquisition Risk Management key process areas (see Resources). The effectiveness and novelty of i-MATE lies in the combination of:

- *A defined process.* This process comprises a straightforward series of well-defined process steps for gathering, ranking, and weighting application requirements for integration middleware.
- *A knowledgebase.* This knowledgebase contains several hundred generic requirements for various classes of COTS middleware products, including those unique to services-based integration implementations.
- *A requirements-analysis tool.* The analysis tool enables rapid assessment, experimentation, and presentation of how the middleware products under evaluation compare against the project requirements.

Let's take a look at i-MATE's unique features that make it highly suitable for evaluating integration technologies in the context of building a services-based integration solution.

The process used in i-MATE is similar to those processes described in proceedings for two international conferences on COTS-based systems and software engineering (see Resources). It defines the series of steps undertaken in i-MATE, the decisions that are made, and the artifacts produced at each stage (see Figure 1). Let's take a look at each stage.

Figure 2 Populating the knowledgebase


Middleware products
In-house lab testing and evaluation
MTE evaluation reports
i-MATE Knowledgebase
Client application requirements
i-MATE evaluation process
Middleware product selections
Proof-of-technology prototypes

Elaborate customer requirements. This first step produces a document that captures the customer's requirements. Because the technology and application problems are complex, we usually find that the overall requirements are not fully understood. Consequently, a number of workshops are held with the application stakeholders to elicit the requirements. The stakeholders involved ideally include both IT and business groups. The resulting document details the business and technical requirements that are specific to the need for integration technology in this services-based integration environment. Each requirement is expressed as a single item that can be evaluated against a specific integration technology.

Augment with generic requirements. This step introduces the i-MATE knowledgebase of over 200 generic, broadly applicable requirements for integration technologies. These requirements augment the set of application-specific requirements with generic integration requirements. The output of this step represents the overall application requirements for services-based integration technology, represented as individually identified requirement points.

Rank overall requirements. Working with the key application stakeholders, the overall set of requirements is ranked. At a coarse level, each requirement is deemed as *mandatory, desirable, low priority/not applicable.* Within each of these categories, importance weightings are assigned to give fine-grain control over requirement rankings, in a fashion similar to that which is described in external resources for COTS products (see Resources). The output of this step is a collection of weighted requirements stored in the i-MATE requirements-analysis tool.

Identify candidate products. This step identifies the three-to-five integration products that are most likely to be applicable to the overall application requirements. In some cases, the customer has already identified a short list, based on both technical and business reasons. In others, we use our experience to work with the customer to identify the most likely candidates.

Product evaluation. In workshops with the key customer stakeholders and product vendor representatives, we evaluate each of the candidate products against the overall requirements. Scores are allocated against each requirement point for each product and captured in the requirements-analysis tool. This involves in-depth technical discussions and step-

SOI-Specific Requirements

Capturing SOI-specific requirements helps with the plan and design for the enterprise-wide SOA, saving both time and effort as well as helping produce a low-risk outcome.

- The product must support the Web Service Basic Profile set of specifications—for example, WSDL, SOAP, UDDI, and XML.
- The product must support advanced WS- specifications (including WS-Security, WS-Coordination, WS-ReliableMessaging, and so on).
- The vendor must implement the WS- set of specifications through toolkit downloads within six months of the specification release date.
- The product must support custom Web services support through exposing SDK APIs or provide intercept mechanisms.
- The product must have inherent Web services support, rather than through a separate bolted-on product.
- Access to various Web services features—for example, security and transactional guarantees—must be supported through both programmatic APIs and declarative means.
- The product must support easy "Web services enablement" of existing business service interfaces.
- The product must interoperate with all applications that expose services through Web services standard interfaces.
- The product must not have proprietary extensions that would break Web services-based interoperability requirements.
- The Web services toolkit must have undergone the WS-Interoperability workshops' rigorous testing to demonstrate interoperability with other Web services toolkits.
- The vendor must be an active member of the WS-I organization.

ping through relevant application scenarios to understand precisely how the integration products behave. In some cases, product capabilities and features can cause the process to iterate and refine the requirement rankings. Once all products have been evaluated, the requirements-analysis tool automatically calculates weighted summary scores based on individual requirement point scores and weightings. Summary charts are also created automatically to support efficient presentation and reporting.

Scenario analysis. By varying requirement weightings, the requirements-analysis tool makes it trivial to explore various what-if scenarios and trade-offs, which can be used to further differentiate between candidate products or confirm the appropriateness of a certain product under varying requirements. The output from this step is the recommendation of one or more products that can satisfy the application requirements (see Table 1).

Building a Prototype

Proof-of-technology prototype. When the outcome from the product evaluation is not 100 percent conclusive, a rapid proof-of-technology prototype is developed. The prototype typically implements one critical scenario that will exercise and/or stress the requirement(s) considered to have highest priority. Even very simple prototypes are powerful tools that give concrete, indisputable evidence of product capabilities. In several i-MATE projects, the results of prototypes have provided the final differentiation required to finalize product selection.

In fact, a prototyping phase is always recommended in i-MATE, even if one product emerges from the process as a clear leader. However, when only one product is considered, the prototyping task is not competitive and can be scoped and structured more toward validating key application requirements.

In terms of resources, the product evaluation and proof-of-technology prototype stages invariably consume most of the effort in i-MATE. Product evaluation takes, on average, between one and three days' effort per product under evaluation, depending on the i-MATE team's familiarity with the particular product. Prototyping is more variable and depends on the complexity of the desired prototype. In most cases a simple system suffices, and the prototyping stage lasts less than one week. In other applications where risks are higher, prototyping has extended to around one month.

The i-MATE knowledgebase contains an extensive set of generic requirements for more generic middleware technologies as well as those specific to ESB technologies. These generic requirements are derived from the practical experiences from CSIRO's Middleware Technology Evaluation project, working with product vendors, and working on consulting engagements for clients (see Resources).

There is a different instantiation of the overall knowledgebase for each class of products because there are different classes of middleware products. For example, the knowledgebase is versioned for services-based integration technologies, EAI, application server technologies, and CORBA technologies. We will look at the services-based integration knowledgebase in the next example.

A detailed analysis of the set of generic requirements has resulted in each knowledgebase being structured as a set of high-level categories, which encapsulate several individual requirement items. Presentation of the whole knowledgebase is beyond the scope of this article, but as an example, the categories for the services-based integration technology version are described here.

ESB high-level evaluation categories. Each high-level category contains typically between 10 and 20 individual requirements that relate to that category. For example, the Web services support category contains this set of individual technology requirements:

Table 1 Analyzing scenarios

Costing	Basic technology/services/training costs
XML message and service management	Facilities available for message format definition, service interface/contract definitions, service management
Integration architecture	Core architectural features, flexibility, eventing services, how are services discovered/integrated/invoked
Adapters	Range and quality of adapters available for integration to external systems
XML translation and data transformation	How capable are the in-built XML translation tools? Processing performance characteristics, data-mapping capabilities
Delivery quality	Fundamentals of service bus, mode of operation, quality of service, (for example, reliable messaging), routing capabilities
Web services support	Compliant to Web services standards? How comprehensive is the Web service support? Is Web services support inherent or bolted on?
Development and support	How are applications developed and debugged?
Performance	Raw performance and scalability issues
Security	Authentication, authorization, encryption, single sign-on facilities
Transaction services	Facilities available for supporting transactional behavior
Workflow	Business process management and automation features, business activity monitoring
System management	How are applications deployed, managed, and versioned?
Technical	Miscellaneous technical requirements

Web services support. These requirement points cover low-level, detailed features of the integration technologies. All services-based integration solutions will inevitably require some or all of these capabilities. During an i-MATE project, the client is led through the contents of the knowledgebase, and the importance of each requirement to the client application is determined. In some projects, the client is technologically cognizant, and the process is fast and straightforward, requiring less than a day to complete. In other projects, the client relies on the i-MATE team to explain the implications of many of the requirements, and their relative importance is set collaboratively.

In addition to the categorized requirements, the i-MATE knowledgebase is populated with evaluations of various versions of major middleware products. Each product in the knowledgebase is ranked on a scale of 1–5 against individual requirements. The rankings occur and are kept current through two mechanisms, as will be explained shortly (see Figure 2). The first is the MTE project, which rigorously evaluates middleware technologies using a defined, repeatable approach. The outputs of the MTE evaluations feed directly into the evaluations in the i-MATE knowledgebase. The second mechanism is the i-MATE projects themselves. Clients often request that an integration technology or other middleware product or version that has not previously been evaluated is assessed during an acquisition project. In such circumstances, the i-MATE team works with the product vendor to rank the product features. The resulting evaluation extends the coverage of the products in the knowledgebase, and these can be reused in subsequent projects.

By reusing the generic requirements in i-MATE, organizations are saved the cost of developing their own set of integration technology requirements. Effort can thus be focused on capturing their application-specific requirements, and planning and designing for the enterprise-wide SOA, which saves both time and effort and helps produce a low-risk outcome (see the sidebar, "SOI-Specific Requirements").

A custom requirements-analysis tool has been built to support trade-off analysis as part of the i-MATE process. The basic tool functionality provides capture of individual requirements points, both generic and application-specific, structured into high-level categories; capture of product rankings and requirement weightings; instantaneous calculation of weighted averages for requirement categories; and instantaneous calculation and reporting of the evaluation outcomes using charts and graphs.

The trade-off analysis tool is based on a spreadsheet program. The major strength of this approach is demonstrated during the product evaluation and what-if scenario analysis phases of i-MATE. As the spreadsheet is *live*, any changes made to category rankings or requirement item weighting are immediately reflected in the graphs depicting the evaluation scores.

On the screen for setting requirement category weightings, in this project the rules engine, development and support, and system management categories are deemed highest priority. These settings generate a set of graphs representing product rankings once the evaluation of the products is complete. At this stage, it is usually desirable to explore how the overall evaluation result may vary if one of these high-priority categories is reduced to a medium level of priority. Changing any of the priority values causes the spreadsheet to instantly reflect these changed priorities in the evaluation results, which makes it feasible to rapidly explore alternatives and confirm the evaluation results under various alternative scenarios.

The i-MATE process has been described as a process to ease the evaluation of integration technologies within the context of implementing an SOA. Integration technologies are complex, highly technical, and diverse collections of products that typically operate in mission-critical business environments. It is also a significant IT investment in ensuring smooth future integration. i-MATE's key contribution in easing the integration technology evaluation process lies in the combination of a prefabricated, reusable set of generic requirements that are based on the analysis of middleware components characteristics; a process for incorporating application-specific requirements, weighting individual requirements; and tool support for capturing and rapidly exploring requirement trade-offs and generating reports that show how the middleware products compare against the requirements.

The services-based integration approach holds the key to seamless integration and interoperability in the future. If things are done correctly, we should not be faced with the traditional EAI problems anymore. With the advent of Web services, and the whole industry contributing and participating in the standardization effort, for the first time in our IT industry, Web services and SOA hold the promise of solving the EAI challenge. Services-based Integration is an important pattern for implementing such a vision. The careful selection of an integration technology for this purpose is absolutely crucial in contributing toward the success of such a software engineering endeavor. •

About the Authors

Anna Liu is an architect in the Microsoft Australia Developer and Platform Evangelism Group. She specializes in EAI projects, is passionate about codifying good software engineering best practices, and accelerates enterprise adoptions of these practices and learning.

Ian Gorton is a senior researcher at National ICT Australia. Until March 2004 he was chief architect in Information Sciences and Engineering at the US Department of Energy's Pacific Northwest National Laboratory, and previously he worked at Microsoft and IBM, as well as in other research positions, including the CSIRO.

Resources

IEEE Computer Society

"A Case Study in Applying a Systematic Method for COTS Selection," in Proceedings of the 18th International Conference on Software Engineering, Jyrki Kontio, pp. 201–209 (March 1996)

"A Formal Process for Evaluating COTS Software Products," Patricia K. Lawlis et al., Vol. 34, No. 5 (May 2001)

"A Process for COTS Software Product Evaluation," in Proceedings of the 1st International Conference on COTS-Based Systems – ICCBSS 2002, S. Comella-Dorda, et al., pp. 86-96 (February 2002)

"Software Acquisition Capability Maturity Model (SA-CMM)" Version 1.03 Technical Report CMU/SEI-2002-TR-010, Jack Cooper and Matt Fisher (March 2002)

"Software Component Quality Assessment in Practice: Successes and Practical Impediments," in Proceedings of the International Conference on Software Engineering, Ian Gorton and Anna Liu, pp. 555–559 (May 2002)

"Streamlining the Acquisition Process for Large-Scale COTS Middleware Components," in Proceedings of the 1st International Conference on COTS-based Software Systems, Ian Gorton and Anna Liu, Vol. 2255, pp. 122–131, Lecture Notes in Computer Science (Springer-Verlag 2002)

Planes, Trains, and Automobiles

by Simon Guest

Summary

Many implementations of Web services exist today across multiple platforms and environments. The majority of these share one thing in common: they all use HTTP as the underlying transport. The ubiquitous nature of HTTP has helped Web services gain the adoption they have today. However, is HTTP a great fit for every problem? Are there application architectures that would benefit from using other transports? What are the advantages and disadvantages of doing so?

As many people know, HTTP has a long history before Web services. It has been the default transport for browsing Web pages since early versions on NCSA Mosaic were released to the world. HTTP as it stands is a pretty good fit for Web services. Being pervasive, it typically works well across firewalls and proxy servers, elements (such as WSDL) are easy to test through HTTP, and many HTTP stacks and servers are available on which to build implementations. If we also go back and look at some of the initial design goals for Web services, we see a lot of momentum around publicly facing Web services. As a result, HTTP makes for a perfect choice.

Despite the pervasiveness of HTTP, however, there are scenarios where it doesn't always fit. In my experience, I've seen these scenarios fall into three categories: asynchronous connections, offline and local applications, and distributed computing. Let's take each of these areas in turn, and examine scenarios where HTTP may not be the perfect match.

Contoso consulting is a fictitious organization that employs nearly 5,000 consultants worldwide. In recent years they have undergone a phenomenal expansion, hiring many staff at the end of the dot-com era. Because of this increase in head count, one of the problems the company now faces is with the submission of timesheets. Every week, each consultant must submit a timesheet so that clients' bills are accurate and timely.

Its current model involves sending a timesheet (created using a template in Microsoft Excel) to the accounts department through e-mail. After the timesheet arrives, a member of the accounts group enters it into their accounting system to record hours to be billed. The accounting system is currently mainframe based.

As you can imagine, this model for submitting timesheets is not scaling well with the company's current expansion. Despite hiring more people for the accounts department, the method of dealing with incoming timesheets through e-mail is becoming laborious because of the manual process of taking the data from Excel into the accounting system.

To help with this system, internal IT has designed a new timesheet submission service. Using Web services, this service will sit in front of the mainframe, accept a timesheet from a consultant, and will automatically enter the details into the accounting system. The Web service design has been kept simple, and a smart client application has been developed for submitting the timesheets (see Figure 1).

One of the design goals early on, however, has been to ensure that the submission of timesheets is performed asynchronously. The accounting system is becoming dated, and the thought of 5,000 consultants all submitting timesheets on the last day of the week is somewhat overwhelming. To overcome this problem, the architect for the system has decided to implement a message queue between the Web service and the accounting system (see Figure 2).

The responsibility of the queue will be to batch requests from the clients before submitting them to the accounting system. This asyn-

Figure 1 A Web services façade is used to expose the Web service.

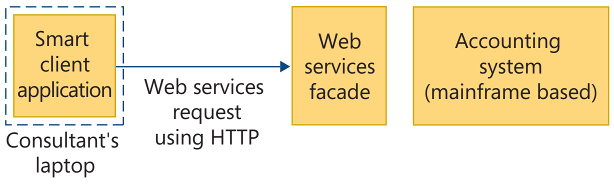

Smart client application
Consultant's laptop
Web services request using HTTP
Web services facade
Accounting system (mainframe based)

Figure 2 A message queue batches requests from clients to the mainframe.

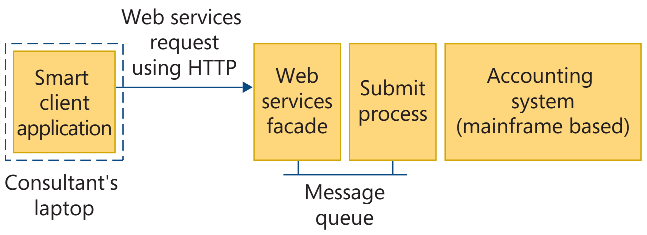


Figure 3 An asynchronous response back to the client is difficult.

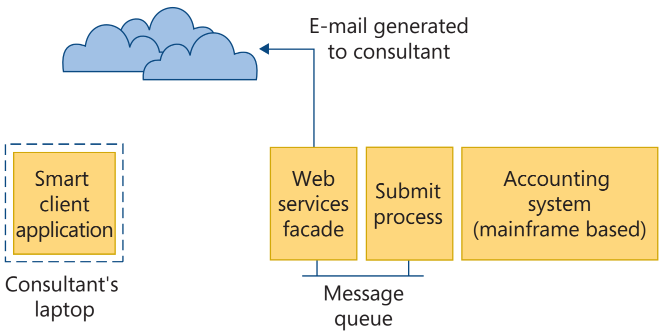


Figure 4 A timesheet is submitted directly to the message queue.

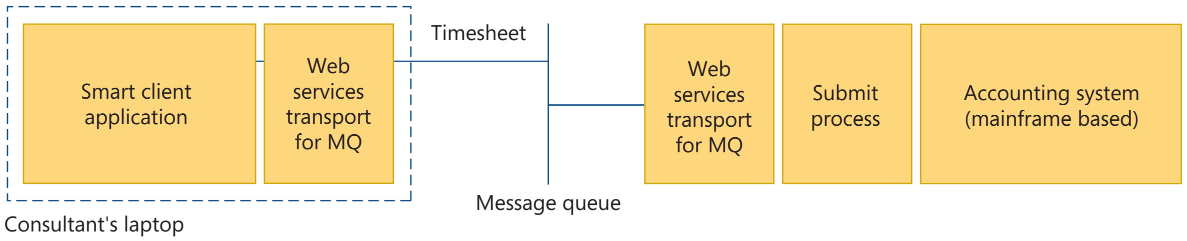
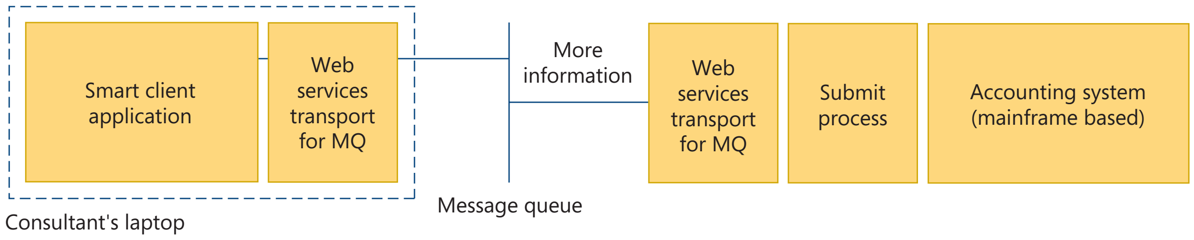

Smart client application
Web services transport for MQ
Timesheet
Web services transport for MQ
Submit process
Accounting system (mainframe based)
Message queue
Consultant's laptop

Figure 5 The accounting system requests more information through the message queue.


Smart client application
Web services transport for MQ
More information
Web services transport for MQ
Submit process
Accounting system (mainframe based)
Message queue
Consultant's laptop

chronous design will avoid overloading the accounting system with too many concurrent requests, and at the same time release the smart client application to do more tasks (that is, the smart client doesn't have to wait until the accounting system has processed the timesheet before moving on to other tasks). This design all looks good in principle, but the IT group notices one potential problem. What happens if there is a problem with the timesheet submission?

Imagine this sequence: A consultant submits a timesheet through the Web service (see Figure 2). Everything works and the timesheet request is placed on the message queue. After a couple of hours (it's Friday evening and the system is busy), the accounting system reads the timesheet from the message queue. In this process it's discovered that the consultant has incorrectly assigned some hours to a project that was previously marked as complete. The accounting system needs this information before it can continue to process the request.

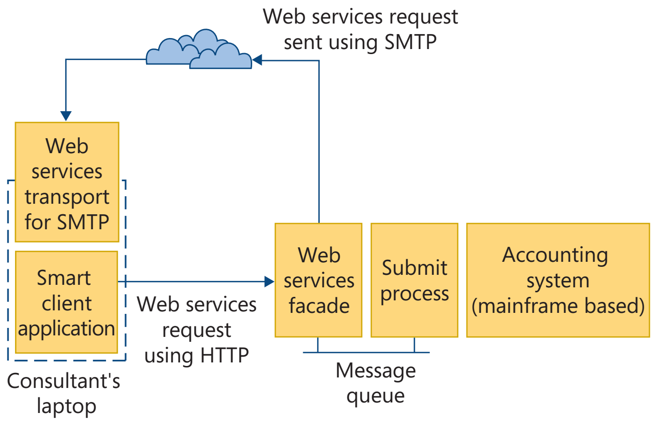
Using the current design, what are the options? The accounting system could raise an alert to a member of the accounting staff (maybe a system message) to indicate that more information is required. The member of the accounting staff could then go chase after the consultant for the correct data. This solution would close the loop, but it's still a manual task, and will only scale so far with the number of people in the organization.

Alternatively, the accounting system could raise an alert to the consultant directly, perhaps sending an e-mail directly to the consultant for the additional information. Again, this solution should close the loop, but the request for information is still disconnected from the original process.

How does the consultant correlate the e-mail to the timesheet submitted? How does the e-mail describe accurately what information is missing? How does the accounting system correlate a new or modified timesheet submission with the old timesheet and the e-mail that was sent? How does the accounting system follow up if the consultant just deletes the e-mail? What happens to the existing submission? There is a possibility this loop will never close.

Let's take a step back and ask why an e-mail was required in the first place? Why couldn't the accounting system just communicate the information directly to the smart client application? The answer: HTTP is a request/response protocol.

Figure 6 In another approach, we use SMTP for the Web services request.


Web services request sent using SMTP
Web services transport for SMTP
Smart client application
Web services request using HTTP
Web services facade
Submit process
Accounting system (mainframe based)
Message queue
Consultant's laptop

Once the timesheet has been submitted and the HTTP request/response has been completed (see Figure 3), it's nearly impossible to communicate back to the client for more information. Even if the client was running a local Web server (to accept incoming Web services requests), what happens if they go offline, or are behind a firewall at a customer's site? How about if their IP address and/or hostname has changed since the last communication? Even more frightening is, who manages the Web server implementations on each of the 5,000 consultants' laptops?

Because HTTP is a request/response protocol, and because it's then very difficult for the service to follow up with the client, alternative asynchronous measures (that is, the e-mail) have to be taken. Unfortunately, because this e-mail is effectively "disconnected" from the original request, it can often take a lot more work to correlate what needs to happen.

To see how we can design a solution using transports other than HTTP, let's look at a couple of alternative approaches.

Web services using a message queue. Coming to the conclusion that HTTP was the culprit was easy. Our first approach for a solution requires thinking about an alternative transport to replace or retrofit the HTTP connections in our design.

Taking our existing design, let's replace the HTTP communication with a message queue. The implementation is unimportant at this stage (it could be MSMQ, IBM MQ Series, Tuxedo, or something else), as long as it's able to reliably handle asynchronous requests.

It's important to note that we are still going to be using Web services—we are very much sending and receiving SOAP messages, except they are being sent using an asynchronous message queue as opposed to HTTP. We can use some kind of Web service-enabled transport for the message queue. So, how does this architecture work with our new scenario?

The smart client submits a timesheet (and a corresponding Web service request is created). This request is placed on the message queue directly instead of sending it over HTTP (see Figure 4). Once the message is placed on the queue, the client can disconnect safely.

Figure 7 A local message queue is used to hold the request offline.

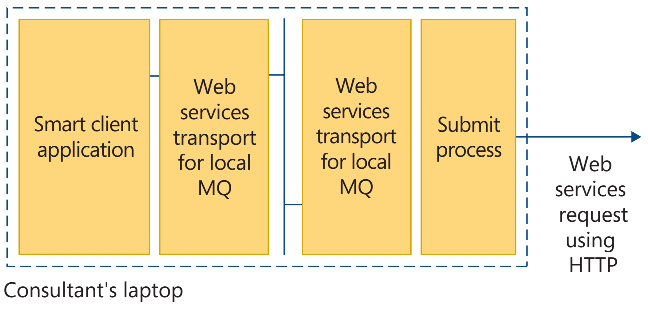

Smart client application
Web services transport for local MQ
Web services transport for local MQ
Submit process
Web services request using HTTP
Consultant's laptop

Figure 8 Two apps on the same machine communicate through HTTP.

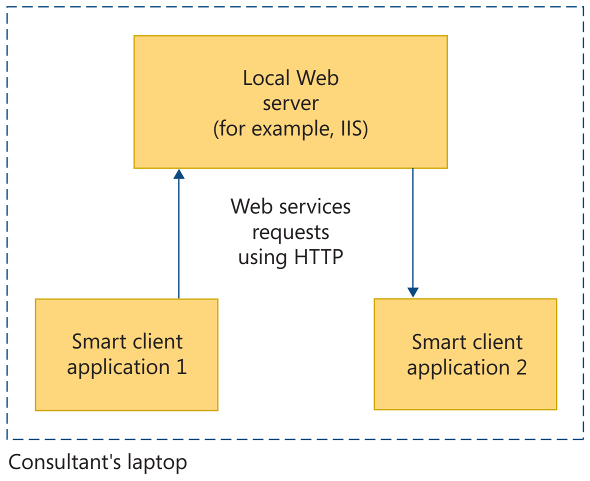
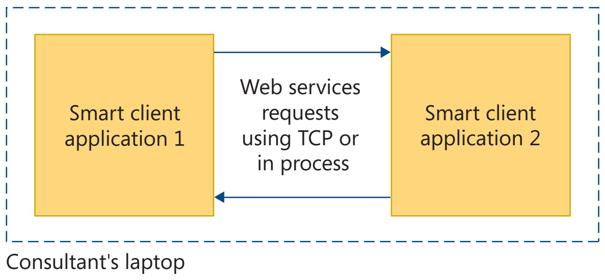

Local Web server (for example, IIS)
Web services requests using HTTP
Smart client application 1
Smart client application 2
Consultant's laptop

Figure 9 TCP or an in-process transport provides a more direct way for local applications to communicate.


Smart client application 1
Web services requests using TCP or in process
Smart client application 2
Consultant's laptop

The server-based Web service will in turn pick up this message, and then communicate with the accounting system to process the request. In the case where the accounting system has to ask for additional information, a new message (a request to the consultant) is placed on the queue: the addressee is the smart client application (see Figure 5).

This message will remain on the queue until the smart client reconnects. To ensure that the message is picked up in a timely fashion, we may consider a background service on the client that connects to the message queue and launches the timesheet application's "missing information" dialog for the consultant.

This approach provides for a solution to our closed-loop issue, and could well offer a more automated approach, but it has one flaw. The smart client must be able to connect to the message queue to process incoming requests from the accounting systems. The majority of message queue vendors do this by accessing some proprietary message queue APIs. How about if the consultant is on the road? How about if the consultant has only Web and e-mail access in an airport? These messages are not going to be picked up until he or she connects to the corporate network, which could be unacceptable. Let's look at a second approach using another transport.

Web services using both HTTP and SMTP. One of the main problems with the original design is that once the accounting system sent the e-mail to the consultant for more information, it effectively created an open loop. This design relies on the consultant having to manually associate the incoming text mail message with the process in the application.

The transport itself, however, is reasonably effective. With the reliability of e-mail these days, it's more than likely that the consultant will have received the e-mail. With this factor in mind, we could consider a new design that builds on this reliability.

Figure 10 A Web services transport is used to log a message to SQL.

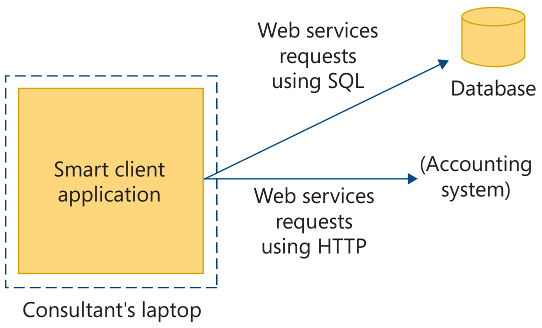
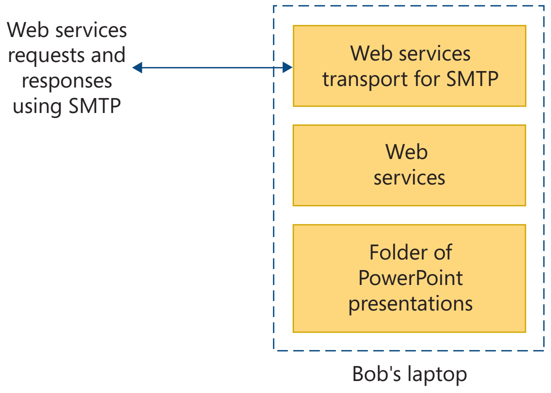


Figure 11 Bob's Web service processes requests and responses over SMTP.


Web services requests and responses using SMTP
Web services transport for SMTP
Web services
Folder of PowerPoint presentations
Bob's laptop

Here, a Web service request over HTTP is still used to submit the timesheet. As with the original design, this request is committed to a message queue to release the smart client connection. In this design, if there is a problem with the timesheet, an e-mail is sent, but not an e-mail to the consultant. Instead, an e-mail message is generated that contains a Web service request for the originating smart client application. What we are doing is initiating a Web service request for the additional information, but using SMTP as the transport (see Figure 6).

The smart client needs a couple of modifications to make this solution work. We need some way of retrieving the SOAP request through e-mail—either through a filter on the consultant's inbox or a separate e-mail account for the smart client application. Secondly, once the e-mail is received, the smart client application must process it and cause the correct action to happen on the client (for example, a dialog asking the consultant for the missing information).

Figure 12 Joe's laptop uses SMTP to send 50 Web services requests.

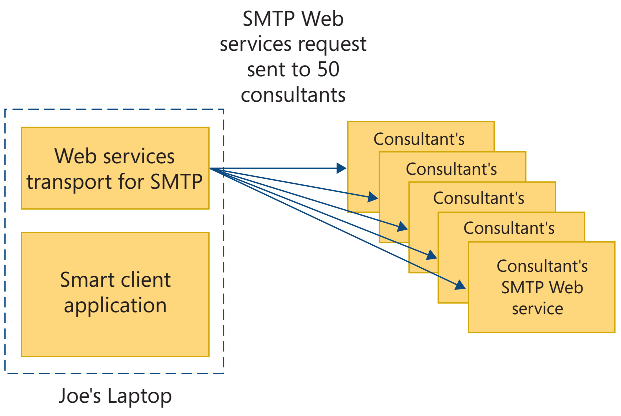
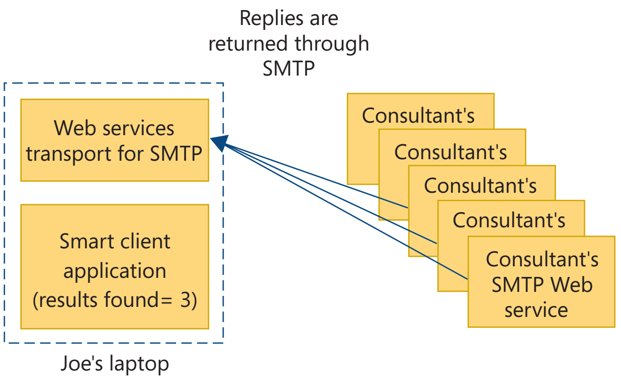

SMTP Web services request sent to 50 consultants
Web services transport for SMTP
Smart client application
Consultant's
Consultant's
Consultant's
Consultant's
Consultant's SMTP Web service
Joe's Laptop

Figure 13 The smart client application processes returned responses.


Replies are returned through SMTP
Web services transport for SMTP
Smart client application (results found= 3)
Consultant's
Consultant's
Consultant's
Consultant's
Consultant's SMTP Web service
Joe's laptop

One advantage to this approach is that it uses existing transports, allows the accounting application to initiate a request to the smart client application, and (providing we can access e-mail from a remote location) does not restrict the consultant to having to connect to the corporate network in which to submit expenses. Remember also that because the request is a Web service, other standards (such as WS-Security) can be applied equally, providing integrity and confidentiality for the message even though it's being sent over public SMTP servers.

This concept of asynchronous connections can also apply equally on the client. Taking our previous example, imagine that the consultant is about to submit a timesheet. He or she generates the timesheet within the smart client application, and then submits it (through HTTP) to the Web service.

This approach works perfectly well, providing there is a connection to the Web service. What happens when the consultant submits the timesheet at a location where there is no connectivity (for example, when he or she is in an airplane at 30,000 feet between customers' sites?). In this instance we would have to think of some kind of offline approach. Upon clicking the Submit button, the design of the smart client would have to detect that there is no network connection, and the operation would be suspended or saved to a local database or queue (see Figure 7).

Another alternative to this approach is to consider a second transport. Instead of using HTTP directly from the smart client application, we could consider a local queuing transport to provide this offline functionality automatically.

Here, a local queue (using for example, MSMQ) is installed on the consultant's laptop. Instead of using HTTP, the SOAP request is placed on the queue by default. A second process, potentially running in the background on the consultant's machine, would monitor the local MSMQ instance for new messages and, on a frequent basis, would check to see whether a connection to the HTTP-based Web service could be established. Once these two can be connected, the message is forwarded between the transports.

For smart client applications, using alternative Web services transports also opens up other options. Imagine that you have two smart client applications running on the same machine that need to communicate (see Figure 8). Initiating calls using Web services over HTTP could be overkill as it would require a local instance of a Web server, and each request would likely traverse the network stack on the machine.

A more efficient way of using two smart client applications would be to use either a TCP (socket)-based transport or an in-process (or shared memory) transport (see Figure 9). In this case, the two applications on

Table 1 When to implement transports over HTTP

Problem when using HTTP	Alternate transport	Advantages	Disadvantages
Difficult to communicate back with the client after a request has been made	Message queue (for example, MSMQ or IBM MQ Series)	True asynchronous connection	Difficult to access in a remote location (requires visibility of the queue)
Difficult to communicate back with the client after a request has been made	SMTP	True asynchronous connection	Requires filter or alternative mailbox to process incoming messages
Need to correctly handle connection state if HTTP service is unavailable	Local message queue	Submit messages even when offline	Requires local install of message queue, plus monitoring process
Logging Web services requests requires additional code	SQL	Submits messages directly to SQL server as an alternative transport	Need code on SQL server to map Web services requests to the database schema
Web server required for two applications on the same machine	TCP or in process	Direct communication without an additional server	Requires management of open sockets on a local machine (for example, pooling, local firewall, and so on)
Difficult to expose peer-to-peer Web services between organizations (unless opening holes in the firewall)	SMTP	Little additional infrastructure required (assuming the e-mail server already exists)	Security difficult to control; only good for asynchronous, potentially long running scenarios

the same machine can communicate using standard Web service requests and responses, but using a lightweight and manageable transport.

In addition, using our timesheet example, how about if we wanted to implement a way of logging all Web services requests (for auditing purposes). We would probably approach this task by creating a log of the message before it leaves for the service, which would involve a filter or class to take the message to the database. This solution works, but an easier approach may be to implement a Web services transport (see Figure 10). A transport could use a SQL database to log outgoing requests, yet to the smart client application it looks like just another transport. Here, the Web services request is sent using two transports. The first goes to the intended recipient (through HTTP). The second is sent to the database for logging using a SQL transport.

A Peer-to-Peer Approach

Finally, another area that has great potential for alternative Web services transports is peer-to-peer computing. Let's take a look at an example. Bob is a consultant at Contoso. On his laptop he has a directory of PowerPoint files that he uses for presentations with customers. This directory travels with him wherever he goes. It's constantly being worked on, and it must work in both online and offline scenarios.

Being a good citizen, Bob wishes to share these PowerPoint slides with his fellow co-workers, both inside the company and with members of other organizations. Many people e-mail him today asking whether he has a particular PowerPoint slide on a topic, and while this process works, searching and replying to these requests are consuming a lot of Bob's time.

Bob is considering building a centralized Web service to host his PowerPoint slides. It should be available to everyone, yet he must be able to access these slides in offline scenarios too. He considers the steps required to implement such a service.

Setting up a central server. Bob is going to have to take his directory of PowerPoint files and host it somewhere centrally. Hosting it centrally will include not only finding enough disk space, but also a consideration for managing backups and updating with the latest versions.

Exposing a Web service. With the server setup, Bob is going to have to install a Web server on the machine, create a Web service, and work with the local IT group to make sure that it is correctly hosted behind Contoso's firewall (probably in the DMZ).

Create a smart client application to access. Bob is thinking of creating a smart client application that will let him keep an offline version of the slides he needs at a moment's notice.

Bob looks at this process—it certainly looks like a lot of work, plus he's uncertain how well it will scale. How about if the other 5,000 consultants in the organization want to do something similar? Will they have to go through the same approach? How about if they are not as technically savvy as Bob?

Bob takes a step back and thinks about why he wants to do this setup. The current system works pretty well; it's just that he gets flooded with too many e-mail requests about presentations that he's recently delivered, or may have. He could potentially create a Web service on his laptop to handle these incoming requests. The Web service could search his directory of PowerPoint files and retrieve certain slides for clients. The problem with this approach, using HTTP, is that Bob's laptop has to be on and accessible for this to work. Generally, Bob is out of the office a lot, and how does he allow access to his laptop through a firewall for external customers? It's looking fairly unmanageable.

Table 2 Implementing alternative transport for Web services

Transport	Description
Indigo	Indigo is the code name for the next-generation, distributed-computing environment from Microsoft. Indigo offers the promise of multiple transports for Web services, together with a unified programming model. Indigo natively supports HTTP, TCP, and MSMQ in the March 2005 CTP. The programming model allows an easy-to-extend interface for other transports.
Web Services Enhancements (WSE)	For those wanting to implement Web services today, alternate transports can also be realized using WSE. WSE provides an API called a custom transport, which allows transports other than HTTP to be used. Custom transports today include samples for MSMQ, IBM MQ Series, SMTP, UDP, SQL Server, Named Pipes, TCP, and in process.
Java Message Service (JMS) API	A number of Java application server vendors are now providing Web services support through JMS. This alternative allows SOAP request and responses to be processed on a JMS queue.
Java API for XML Mail (JAXMail)	JAXMail (part of Sun JWSDP) is a JAX-RPC (Java Web services) extension to provide support for the SMTP protocol.

After reading about using alternate transports for Web services, Bob comes up with a new design. He is going to create a Web service for his laptop, but instead of accepting incoming connections over HTTP, he will use SMTP (e-mail) instead (see Figure 11). Clients can send him Web services requests to search and retrieve his local store of PowerPoint files. To do this, Bob will create a small smart client application that generates these requests.

The beauty of this design is that Bob and others can now take advantage of the distributed functionality that e-mail provides. Bob shares his new Web service application with 50 of the other consultants at Contoso (see Figure 12). What we now have is a very dynamic way of using Web services to look up PowerPoint files that are held locally on a number of machines.

For example, Joe is looking for a PowerPoint presentation on the topic of C#. He enters the query "C#" into a smart client application to create a Web services request that is sent through SMTP to an e-mail distribution list that contains the 50 consultants running Bob's Web service. Once the message is received, the Web service running on each of these laptops performs a search based on Joe's criteria. The list of results is then sent back to Joe's calling application, which can display them as responses are received (again through SMTP).

Joe can now start searching the results as they come back (remember, just like e-mail he doesn't need everyone to reply; just enough people that have the PowerPoint slide he is looking for have to). When he finds the correct one, a similar request, using Web services over SMTP, can be made to acquire the presentation (see Figure 13).

The approaches we've discussed here may raise more questions than they provide answers for. Hopefully, you can see that using Web services with alternative transports can open up a new range of applications that have until now been restricted by the use of HTTP.

One of the questions that you may have is "When should I implement a transport other than HTTP?" To help answer that question, and to summarize the scenarios discussed here, refer to Table 1. Although relatively a new area, significant progress is being made around implementing alternative transports for Web services (see Table 2). •

About the Author

Simon Guest is a program manager in the Architecture Strategy team at Microsoft Corporation and specializes in interoperability and integration. Simon is the author of *The Microsoft .NET and J2EE Interoperability Toolkit* (Microsoft Press, September 2003).

Enable Internet-Scale Computing

by Savas Parastatidis and Jim Webber

Summary

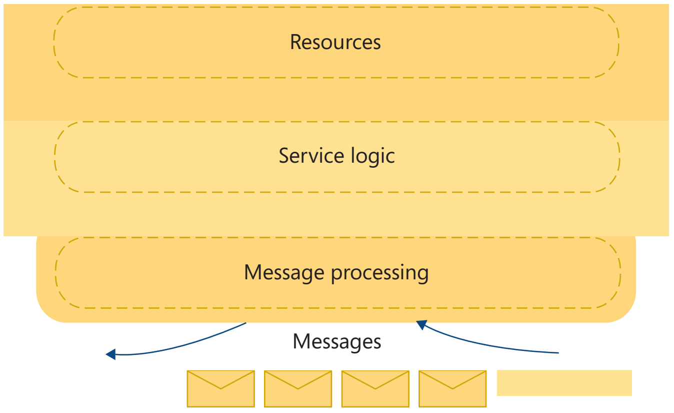
High-performance computing (HPC) has evolved from a discipline solely concerned with efficient execution of code on parallel architectures to be more closely aligned with the field of distributed systems. Modern HPC is as much concerned with access to data and specialized devices in wide-area networks (WANs) as much as it is with crunching numbers as quickly as possible. The focus of HPC has shifted toward enabling the transparent and most efficient utilization of a wide range of capabilities made available over networks, as seamlessly as the way in which an electrical grid delivers electricity. Such a vision requires significant intellectual and architectural investment. Explore a service-oriented approach for enabling Internet-scale, high-performance applications.

From networks for workstations to the Internet, the high-performance computing community has long advocated composing individual computing resources in an attempt to provide higher quality of service (for example, in terms of processing time, size of data store, bandwidth and latency, remote instrument access, and special algorithm integration). In recent years this progression has been driven by the vision of "grid computing" where the computational, storage power, and specialist functionality of arbitrary networked devices is to be made available on demand to any other connected device that is allowed to access them.

Concurrently, the distributed systems community has been working on design principles and technologies for Internet-scale integration (for example, Web and Web services). Recently the term service-oriented architecture (SOA) has emerged as a popular piece of terminology, in some part because of the hype surrounding the introduction of Web services. While Web services are perceived as an enabling technology for building service-oriented applications, they should be treated as an implementation technology of the set of principles that constitute service orientation. The promise of SOA and Web services is the enabling of loose coupling, robustness, scalability, extensibility, and interoperability. These are precisely the features required of a global fabric for grid computing—a popular buzzword used to refer to distributed, high-performance computing (HPC) or Internet-scale computing.

Here, we describe grid computing and service orientation, and we'll discuss how high-performance applications can be designed, deployed, and maintained by using message-orientation and protocol-based integration. We'll also present our approach on how large-scale HPC applications can work with a multitude of resources and state in a manner that is consistent with SOA principles.

Figure 1 The archetypal structure of a service



Grid Computing

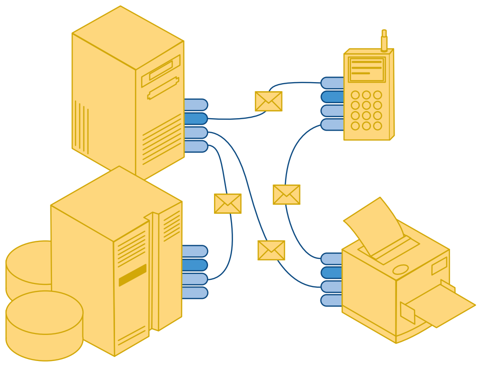
Grid computing is overloaded and has different meanings to different communities (and vendors). Some of the common interpretations are on-demand computing; utility computing; seamless computing; supercomputer interconnectivity; virtual worldwide computer; SETI@home and ClimatePrediction.net; BOINC-style projects (see Resources); and virtual organizations.

We adopt the view that grid computing is synonymous to Internet-scale computing with a focus on the dynamic exploitation of distributed resources for HPC. When building grid infrastructure and applications we promote the application of the same principles, techniques, and technologies that are typical of modern distributed systems practice, with service orientation as the architectural paradigm of choice and Web services as the implementation technology.

While service orientation is not a new architectural paradigm, the advent of Web services has reinvigorated interest in the approach. It is, however, a misconception that Web services are a form of software magic that somehow automatically corrals the architect toward a loosely-coupled solution that is scalable, robust, and dependable. Certainly, it is possible (and generally highly desirable) to build service-oriented applications using Web services protocols and toolkits; however, it is equally possible to build applications that violate every architectural principle and tenet of SOA.

As researchers and developers have re-branded their work to be in vogue with the latest buzzwords, SOA has become diluted and impre-

Figure 2 Networked applications are built through the exchange of messages between services hosted in devices.



cise. Lacking a widely accepted definition of a service, we propose that a service is the logical manifestation of some physical or logical resources (for example, databases, programs, devices, humans, and so on) and/or some application logic that is exposed to the network. And, services interact by exchanging messages.

Services consist of some resources (for example, data, programs, or devices); service logic; and a message-processing layer that works with message exchanges (see Figure 1). Messages arrive at the service and are acted on by the service logic, utilizing the service's resources (if any) as required. Service implementations may be of any scale—from a single operating system process to enterprise-wide business processes.

Services may be hosted on devices of arbitrary capability (for example, workstations, databases, printers, phones, and personal digital assistants) providing different types of functionality to a network-based application. This functionality promotes the concept of a connected world in which no single device and/or service is isolated. Interesting applications are built through the composition of services and the exchange of messages (see Figure 2).

A *message* is the unit of communication between services. Service-oriented systems do not expose abstractions like classes, objects, methods, and remote procedures, but are instead based around the concept of message transfer. Of course, single message transfers have limited utility, so there is a tendency for a number of message transfers to be grouped logically to form message exchange patterns (MEPs)—for example, an incoming and an outgoing message that are related can form a "requestresponse" MEP—to support richer interactions. MEPs are grouped to form protocols that capture the messaging behavior of a service (sometimes known as a conversation) for a specific interaction. Such protocols may be described subsequently in contracts and published to aid integration with the service (for example, in WSDL or SSDL2).

Protocols and Contracts

The behavior of a service in a distributed application is captured through the set of protocols that it supports. The notion of *protocol* is a departure from the traditional object-oriented world where behavioral semantics are associated with types, exposed through methods, and coupled with particular end points (the point of access for particular instances). Instead, a protocol describes the externally visible behavior of a service only in terms of the messages, message exchange patterns, and ordering of those MEPs that are supported by the service.

Protocols are usually described through contracts to which services adhere. A contract is a description of the policy (for example, security requirements or encryption capabilities) and quality-of-service characteristics (for example, support for transactions), which a service supports and/or requires, in addition to the set of messages and MEPs that convey functional information to and from the service.

Many organizations are realizing the cost benefits from using clusters of workstations as alternative platforms to specialized supercomputer facilities for their HPC needs. Until recently, such cluster-based solutions have been treated as dedicated computational and/or storage resources. Enterprises are now seeking to gain in terms of both lower cost and performance by using the idle processing power, distributed storage capacity, and other capabilities available by their deployed workstation-based infrastructure—an approach commonly referred to as "intra-enterprise grid computing."

Here we explore dedicated clusters and how service orientation can be used for building such solutions before proposing an approach to building intra-enterprise, distributed, high-performance architectures.

Purpose-built commodity, hardware-based solutions for HPC are not uncommon inside an administrative domain of organizations with requirements for high-performance computation. Such solutions are usually implemented by one or more clusters of workstations with high-speed interconnects (for example, Myrinet, SCI, Gigabit Ethernet, and so forth). Some such solutions attempt to provide a single-computer image to applications through the implementation—in hardware or software—of techniques that hide the distribution of CPUs, memory, and storage.

Developers are presented with a familiar programming abstraction, that of shared-memory symmetric multiprocessing. However, such approaches have a tendency to limit scalability of computational nodes, which may become an issue for certain types of parallel applications. Specialized message-oriented middleware solutions (for example, MPI) are usually employed to work with the problem of scalability but at the cost of requiring explicit management of the degree of parallelism by the application. There is a lot of work in the parallel computing literature that discusses the advantages and disadvantages of the shared-memory versus message-passing paradigms for parallel applications.

Dedicated clusters for HPC are usually considered and managed as single resources. To enable better utilization of such resources, a service-based approach is preferable. For example, access to resources is usually controlled by a job submission, queuing, and scheduling service that ensures optimal system utilization. Web services technologies can be used for the implementation of such services, and indeed benefit from the significant investment in tooling, efficient run-time support, documentation, and user education in the Web services area. The composable nature of Web services technologies makes it easy

Figure 3 Integrating enterprise resources to meet the high-performance requirements of applications

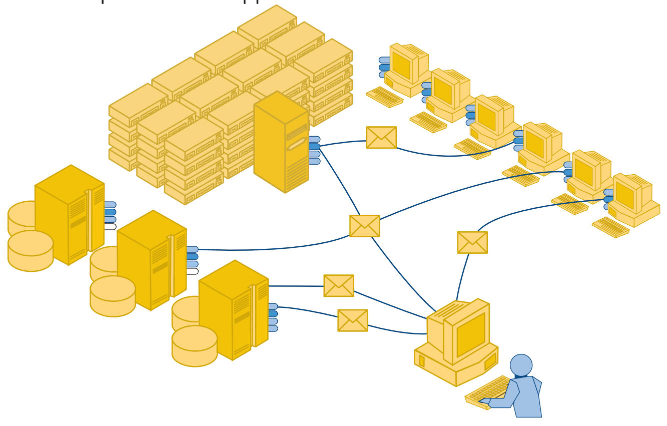
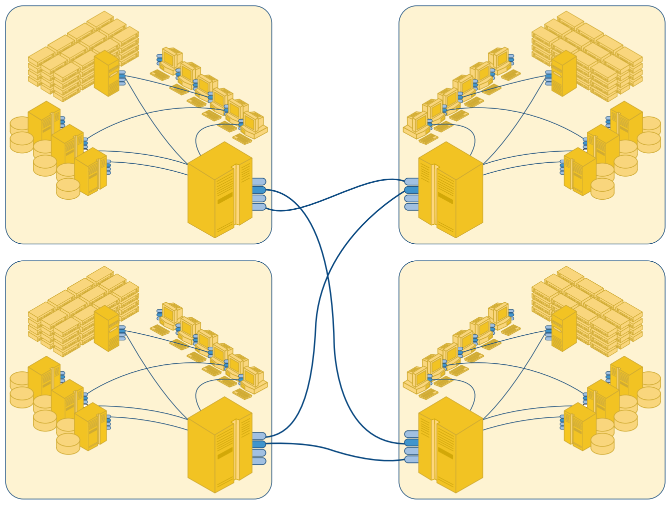


Figure 4 An example of an intra-enterprise SOA with different parts of the enterprise being represented as services



for quality-of-service, nonfunctional needs of the implementation (for example, reliable messaging, security, and transactions) to be incorporated more easily into a heterogeneous environment.

Stealing Cycles from Workstations

An approach that has gained significant momentum recently is the deployment of cycle-stealing technologies implemented by specialized middleware, like Condor (see Resources). Such middleware enables the distribution and management of computational jobs on idle workstations, while allowing a workstation to almost instantly be reclaimed by its console-based user when he or she starts to use the computer, as the cycle-stealing gets suspended, killed, or migrated to another workstation. However, most current implementations of middleware software supporting such installations do not yet leverage interoperable and composable quality-of-service protocols. As a result, it becomes difficult to create interoperable and seamless solutions for HPC within the enterprise.

In intra-enterprise HPC each workstation, database management system, device, and so on exposes some functionality as a service. Building such middleware using these principles, service orientation can result in deployments that can scale to thousands of workstations. A service-oriented approach may increase the flexibility, manageability, and value of such solutions since a large set of widely accepted units of functionality/behavior, made available as protocols, can be leveraged (for example, security, transactions, reliable messaging, and orchestration). Indeed, there are efforts to do just that with existing systems—for example, Condor BirdBath (see Resources).

Future intra-enterprise grid installations will be built around standard services provided by the underlying operating systems. Application protocols like WS-Eventing and WS-Management will be implemented and provided as standard that grid-like solutions can be easily implemented and deployed. We describe an example of a conceptual approach to intra-enterprise, HPC computing using Web services (see the sidebar, "Windows-Based HPC Computing" and Figure 3).

We identify a nonexhaustive set of generic services, functionalities, and features that may be offered and/or supported by each device on the network (see Table 1). Of course, application domain-specific functionalities will also have to be supported (for example, a BLAST service installed on a powerful server to perform bioinformatics analysis, or a service implementing an estimation algorithm for petroleum usage).

Larger enterprises may not be interested in only deploying just single cluster solutions or simply reclaiming the idle processing power of parts of their organizational infrastructure. Instead, they may wish to focus on the encapsulation of entire sets of computational resources behind high-level services that, when composed together, can enable a level of integration that was previously difficult and time-consuming because of the different number of deployed technologies.

The approach to architecting intra-enterprise, high-performance solutions is similar to the approach when the focus is on building the cluster-based solutions discussed earlier (that is, the issues of scalability, loose coupling, and composability apply equally). Quality-of-service protocols like security (for authentication, authorization, and accounting); transactions; reliable messaging; and notifications are all part of the underlying infrastructure and can be used unmodified no matter the type of solution implemented. Moreover, the set of services used for intra-enterprise solutions can also be used unmodified (for example, user-credential management, systems management, application and services deployment, workflow support, data storage and archiving, messaging, and so on).

As mentioned previously, there is still a need for services to offer access to computational and data resources, scheduling implementation, visualization, specialized algorithm functionality, and so forth, depending on the type of application being implemented. This time, however, the services are at a higher level of abstraction because entire collections of resources are encapsulated (see Figure 4).

Standard Integration

We note that even though the complexity of the services has increased from those that we used when building a cluster solution, the complex-

Windows-Based HPC Computing

A set of Windows-based workstations used by staff are part of the underutilized-CPUs Active Directory domain. The enterprise wishes to leverage the computational capabilities of these workstations during their idle period (for example, overnight). The domain administrator pushes the .NET implementation of a set of Web services that provide submission, monitoring, and management of jobs out to workstations through Active Directory. Those services leverage the underlying Web services middleware platform (for example, Indigo) for their security and quality-of-service features (for example, notification, reliable messaging, transactions, and so on) requirements.

Only users that belong to the underutilized-CPUs domain are allowed to submit jobs. WS-Security is used for the authentication and message-encryption requirements with the Kerberos tickets retrieved from the Active Directory. In addition to the workstations, there is also a dedicated cluster for the enterprise's high-performance requirements and a datacenter. Web services installed on these resources expose computational and data storage functionalities to the network. The enterprise's applications are written in a way that can dynamically discover and utilize any distributed computational resources within the enterprise. Therefore, as soon as the functionality is enabled, the computationally intensive applications can automatically start to take advantage of the distributed infrastructure. The users of such applications are unaware of the resources used. •

ity of the architecture has not and the principles and guidelines remain the same. Our distributed application still binds to messages being exchanged and no assumptions are made about an intra-enterprise-wide understanding of interfaces and behaviors of the various components. Architects design applications through the description of messages and the definition of protocols, which capture service behavior.

We observe that as the granularity of the services increases, the need to increase the granularity of the message exchanges is higher. The network is expensive and so architects need to design their protocols and their messages appropriately. As the degree of distribution of an application increases, the need for loose coupling also increases. While in a single cluster or in smaller enterprise environments complete control of the infrastructure and the set of deployed technologies is possible, in an enterprise-wide (or larger) solution it is imperative that integration happens through standard protocols.

Also, it is clear that as the complexity of the scale of an application increases, the functionality of its services becomes even more abstract. From services that expose specific functionality to the network (for example, remote process execution and workstation management) or provide access to a resource (for example, database system and file system), we move to services that support aggregation of functionalities (for example, job queues, message queues, and cluster management) or resource aggregation (for example, storage area network and database federation).

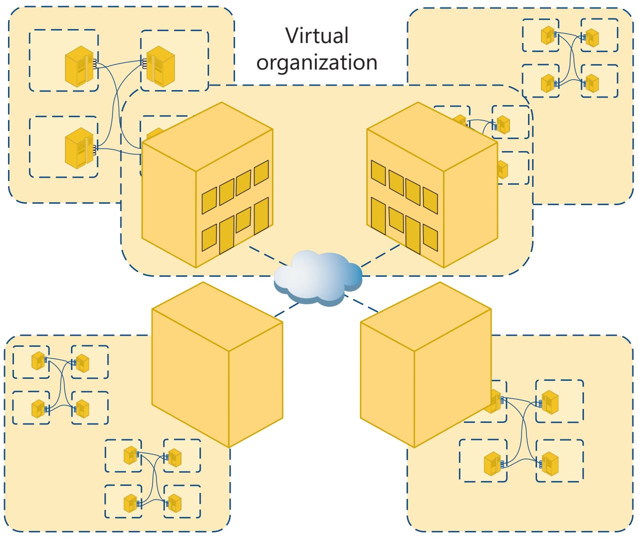
Since the interactions between the services become coarser to minimize the effect of the communication costs between the different parts of the applications and the services become more abstract and coarse grained with respect to the resources they encapsulate, we must design applications using larger building blocks. The larger the scale of a distributed application, the more important it is to devise declarative, protocol-based, and coarse-grained mechanisms for describing behavior. It is at this stage that workflows, contracts, and policies become even more significant. Service orchestration and abstract business processes become necessary, and so relevant technologies like WS-BPEL become an important part of the architect's toolset.

In the same way that no single device is an inaccessible island within an administrative domain, enterprises and organizations are similarly not isolated. As is the case with our physical world, enterprises do businesses with one another, organizations interact, and government institutions collaborate. Services and interactions are fundamental to our day-to-day activities (for example, the banking service, the post office service, and a travel agent service). It is only natural that when we model these activities in a computerized world, we follow a similar architectural approach to the one adopted in the real world.

As one would expect, when it comes to very large-scale applications with a focus on delivering high performance, the nature of the application may lead us to different designs with different strategies in mind compared to an intra-enterprise situation. Like the real world, contracts and service-level agreements (SLAs) are put in place to govern the interactions between enterprises. Virtual organizations may be established—in the same way alliances are formed between enterprises in the physical world—to meet the high-performance needs of the participating entities' applications. Indeed, a distributed, high-performance application may reflect a real-world alliance between enterprises (see Figure 5)—for example, a number of research institutes joining forces to solve a large scientific problem.

For the virtual organizations to be viable, issues such as digital representations of agreements, contract negotiations, nonrepudiation of

Figure 5 An illustration of how enterprises can be joined



interactions, federation of user credentials, policies, and agreed quality-of-service provisioning have to be addressed. High-level, workflow-based descriptions may be put in place to represent the behavior of the virtual organization or to choreograph cross-enterprise business interactions. Applications in this space, when appropriately designed and implemented, may become truly Internet scale.

While the type of services that are found inside the enterprise—like job queuing, scheduling, resource brokering, data access and integration, and visualization—may still be necessary, when we move to Internet scale, care must be taken on how such services are implemented and deployed. Centralized solutions (for example, a single data store service) or tight-coupling behaviors (for example, long-lived transactions across organizations or direct exposure of state) should be avoided.

Of course, one may argue Google and Amazon are spectacular examples of centralized repositories. This observation is true, but it is also the case that these are logical repositories that already use scalable solutions for their implementations; they are already built on top of distributed, replicated, loosely-synchronized data centers. Google and Amazon can be seen as good examples of virtualized data access services that have been designed and implemented with scalability and performance in mind.

With Internet-scale applications, alliances and collaborations between organizations are formed using digital contracts. Such contracts represent the set of SLAs that must be put in place for computational jobs to travel from one organization to the other; for data sources to become visible; for the functionality of special equipment to become available to the partners; for the level of trust on agreed user roles, security, and policy requirements; and so on. Virtual organizations need contracts to govern their operations as it is the case with any collaboration between enterprises in the real world. The contracts are read, validated, and executed by specialized supporting middleware.

Furthermore, in a digital world where every kind of resource is accessible, application requirements and service offerings are described using declarative languages. The supporting middleware is responsible for dynamically matching an application's requirements with a service offering. If necessary, dynamic negotiation of payment and SLAs may have to take place.

For example, an application may advertise that it needs a service offering computational resources with specific hardware and software requirements, a data storage facility of a certain size, a visualization engine with a specific response time, and an equation-solving service with a guaranteed uptime. These requirements are expressed using an XML vocabulary. The resulting document is sent to a (distributed) registry, and a set of available services are discovered. The underlying middleware negotiates the SLA and payment details with the resulting services according to the application's requirements within the set of limits that the end user has set. Once agreement has been set, all parties involved sign a digital contract, which can be used for future disputes.

The SETI@home, ClimatePrediction.net, and other similar projects have demonstrated that through community networks it is possible to bring together resources to solve large problems. If we ignore the controversy surrounding the use of peer-to-peer (P2P) technologies for file sharing, the grid computing promise of collaboration on scientific and business problems is a good match for the capabilities of P2P technologies or community networks.

Future HPC Internet applications should consider P2P technologies as enabling technologies for file transfers and sharing, resource discovery, computation distribution, and so on. For example, we can imagine a P2P network that allows jobs to be submitted and suitable resource for execution to be discovered automatically. P2P networks may be implemented using the same set of Web services technologies to leverage the huge investment in the underlying quality-of-service protocols.

Table 1 Services, functionalities, and features that devices may support

Feature	Description
Security	All aspects of security (for example, authentication, authorization, auditing, confidentiality, and privacy) are worked with by using interoperable and composable protocols like WS-Security, Liberty-Alliance, SAML, XACML, and so on. Identity management and federation solutions also need to be in place (for example, Active Directory).
Job management	Those resources on the network that are capable of hosting jobs for execution offer a job-management service.
Scheduling	Resource utilization information is gathered and then used in the decision process of distributing jobs to the available resources.
Data access	Those resources on the network that provide access to data stores (for example, relational database systems, file systems, and storage area networks) need to expose appropriate services.
Resource registries and monitoring	Either P2P or centralized solutions need to be deployed to allow discovery and monitoring of the devices on the network and their statuses.
Device management	The devices on the network may have to be remotely managed as a collection or individually (for example, Active Directory and WS-Management).

Although the grid computing concept emerged from the supercomputer community, businesses are now also realizing its commercial value. Per-pay or subscription-based access to resources (especially high-performance compute resources) is starting to emerge as a viable business model with large companies already deploying the enabling technologies and services. Of course, the integration of such deployments into applications has to become ubiquitous and completely transparent to the end users for the vision of "utility computing" or "computing as a service" to become a reality.

This realization yields a number of valuable opportunities. Obviously, there will be those companies that will be able to reap the benefits from hosting cost-effective compute resources for others to integrate into their environments on an ad hoc basis. The reciprocal of this situation is that there will be opportunities for companies to more effectively plan their spending on IT infrastructure and decide whether up-front capital investments may be superseded by the pay-as-you-compute model (or not), in addition to the general business agility that moving to an SOA will yield.

The set of typical services/functionalities presented previously (see Table 1) are also needed in Internet-scale, HPC applications. However, they are more abstract and must make different assumptions about the environment in which they are deployed. For example, the issue of who is allowed to send jobs for execution in an organization's compute center will be defined through digital contracts, while the quality of service that each interaction will receive (for example, CPU and data storage allocation) will be controlled by the SLAs defined in the same contract. In addition to the information presented in Table 1, however, we also observe the set of typical services and behaviors for Internet-scale HPC (see Table 2).

Having discussed at a very high and abstract level the architecture of service-oriented, high-performance, distributed applications that can scale across the Internet, let's now touch on some important design and implementation considerations.

Despite the ever-increasing improvements in network latency and bandwidth, communication over commodity network infrastructures is orders of magnitude less efficient than over specialized interconnects or memory-bus architectures. Consequently, care must be taken when architecting, designing, and building HPC-distributed applications so as to minimize the costs associated with message exchanges between components of an application.

The Tenets of SOA

In addition to network costs, the HPC community is also concerned with the computational costs incurred from the processing of XML.

Table 2 Characteristic services, functionalities, and features for Internet-scale, HPC computing

Feature	Description
Brokering	Services to act as brokers for other services will be deployed to enable dynamic discovery of resources or aggregate resources to provide better value.
Payment	A common infrastructure for payment, similar to that currently used for credit card payments in the real world, will become necessary.
Computational/storage logical services	Services to offer access to computational and storage resources will become available, even if those resources do not belong to a single entity, in the same way companies offering electricity exist in the real world.
Global and domain-specific service and resource registries	Global directories like Google will be necessary for the location of resources and other services on the grid. Application domains may deploy their own specialized registries as a way to add value to its domain users (for example, a registry of bioinformatics-related services).
Data transfer	When large datasets need to be transferred across the Internet, specialized, high-performance transfer technologies need to be put in place. The negotiation of which transfer technology is going to be used will take place over standard protocols. P2P technologies could also be employed.
Contracts and policies	Vocabularies and middleware software to create, negotiate, execute, and monitor contracts and policies are going to be vital in an environment where dynamic virtual organizations are formed.
Orchestration	As services are made available around the Internet, technologies to orchestrate them and combine them in application-specific ways are necessary.
Semantics-related technologies	In an environment where a vast number of resources and services are available, reasoning about the available information in a universal way is going to be extremely important.

However, this is an aspect that is being addressed by the SOAP community. In fact, good SOAP implementations already approach the performance of binary mechanisms for short messages (see Resources online at www.ftponline.com), which implies that eventually the limiting factor for message transmission either in binary or SOAP format will be the latency and bandwidth of the network. While we do not wish to denigrate the importance of low latency and high throughput for HPC applications, it is clear that the laggard label that SOAP has attracted is somewhat undeserved.

Loose coupling and scalability are the results of principled design and sensible software architecture. We recommend adopting these tenets for building SOAs:

- The collection of protocols supported by a service determines its behavioral semantics.
- Services bind to messages and the information conveyed though them and not to particular end points and state.
- Messages exchanged between services are self-descriptive (that is, as in REST) insofar as they carry sufficient information to enable a recipient to establish a processing context and the information needed to execute the desired action.
- Services are implemented and evolve independently of one another.
- Integration of services takes place through contract-based agreements.

In addition to these principles, we also promote this set of guidelines when building service-oriented systems:

Statelessness – This property relates to the self-descriptive nature principle mentioned previously. Services should aim to exchange messages that convey all the necessary information necessary for receiving services to re-establish the context of an interaction in a multimessage conversation. Stateless services are easy to scale and make failover fault tolerance very simple.

Rich messages – Communication costs are usually high, hence, aim for protocols that involve rich messages that result in coarse-grained interactions, effectively minimizing the number of times a service has to reach across the network.

State management – As per the traditional *n*-tier application design, service implementation should delegate all aspects of state management to dedicated and specialized data stores (see Figure 1).

Message dispatching – There should be no assumptions about the dispatching mechanisms employed by services. As a result, no dispatching-specific information should leak from the service implementations, across the service boundaries, and conveyed through the message contents (for example, SOAP-RPC, RPC-style SOAP, Document-Wrapped-style WSDL, or method names conveyed as soap:action or wsa:action attributes).

Role-specific coupling – Architects should keep in mind that in an SOA there are no actors like *consumer*, *provider*, *client*, *server*, and so on. These are roles that exist at the application level and not at the building blocks of the architecture: the services. In SOAs, there are only services that exchange messages. Treating a pair of services as client/server introduces a form of coupling that may ultimately be difficult to break.

Services are a sensible abstraction for encapsulating and managing the increasing level of complexity in distributed applications. The beauty of service orientation is that the architectural principles and the guidelines are consistent from an operating system process through to a service that encapsulates an entire business process or even an entire organization. The architectural requirements of high performance, Internet scale, or grid computing are not different from those of enterprise, business-focused computing, and therefore identical principles and guidelines should be used.

The authors would like to thank Paul Watson, professor, School of Computing Science, University of Newcastle upon Tyne, for his useful feedback during the preparation of this article. •

About the Authors

Savas Parastatidis is chief software architect at the School of Computing Science, University of Newcastle upon Tyne.

Jim Webber is a senior developer with ThoughtWorks Australia.

Resources

Architectural Styles and the Design of Network-Based Software Architectures
A doctoral dissertation, University of California, Irvine
www.ics.uci.edu/~fielding/pubs/dissertation/rest_arch_style.htm
"Chapter 5: Representational State Transfer (REST)," Roy Thomas Fielding (2000)

BOINC
http://boinc.berkeley.edu
Berkeley Open Infrastructure for Network Computing

Condor High Throughput Computing
www.cs.wisc.edu/condor/
The Condor Project
www.cs.wisc.edu/condor/birdbath/
BirdBath

The Globus Toolkit
www.globus.org/toolkit/
Welcome to the Globus Toolkit

GridForge
http://forge.gridforum.org/projects/ogsa-wg
Open Grid Services Architecture Working Group (OGSA-WG)

NEReSC
www.neresc.ac.uk/ws-gaf
North-East Regional e-Science Centre, School of Computing Science, University of Newcastle upon Tyne

OASISOpen.org
www.oasisopen.org/committees/tc_cat.php?cat=ws
OASIS Web services

OGSA-DAI
www.ogsadai.org.uk
Open Grid Services Architecture Data Access and Integration

SSDL
http://ssdl.org The SOAP Service Description Language

http://mercury.it.swin.edu.au/ctg/AWSA04/Papers/ng.pdf
"An Evaluation of Contemporary Commercial SOAP Implementations"
Alex Ng, Shiping Chen, and Paul Greenfield (2004)

World Wide Web Consortium
Technology and Society Domain
www.w3.org/2001/sw/
Semantic Web Activity

Product Strategy and Architecture

by Charles Alfred

Summary

Systems exist to generate value for their stakeholders. Unfortunately, this ideal is often met only to a limited degree. Current development methods, such as waterfall, spiral, and agile often provide incomplete and inadequate direction to stakeholders, architects, and developers. Two essential concepts—value models and architecture strategy—are missing from many development processes; however, these concepts can be integrated effectively using the waterfall, spiral, or agile methods.

Creating well-defined value models provides direction that improves the quality of trade-off decisions, especially in systems that are deployed to many users in various settings. The existence of a clearly stated architecture strategy provides a coherent high-level direction for the system in the same way that the United States Constitution does for the U.S. Let's look at how these two concepts can be integrated effectively with the waterfall, spiral, or agile methods.

Requirements can be ineffective compasses. Current ways of building complex software-intensive systems are ineffective, which is not the same as saying that they are inadequate. Many systems built using the waterfall, spiral, or agile methods are deployed successfully and are able to satisfy their stakeholders. However, many are not, and for reasons that are correctable.

Traditional processes for building software-intensive systems, like the waterfall and spiral methods, rely on requirements to provide direction. A common misconception is that requirements are statements that describe the problem. According to Greenfield et al., they aren't (see Resources). They define the solution from the perspective of the users and system sponsor. Requirements have some notable shortcomings:

- *Requirements typically use a binary structure.* They function like pass/fail grades in a college course, and provide little if any help in making trade-off decisions. Of course, these trade-off decisions must get made at some point in the process. Often they are made implicitly, and without full consideration of the implications.
- *Requirements are used frequently as the basis for specifying testable acceptance criteria for a system.* In the process of making them specific, important design decisions are made implicitly, without full considerations of the implications. Eventually, these decisions must be reversed at a significant cost, or they end up limiting the potential of the system.
- *Requirements tend to treat all individuals of a given user-type the same.* For example, use-case scenarios for a medical system might refer to physicians and nurses, while those for a real estate system might refer to buyer, seller, agent, and lender. The problem is two physicians aren't the same, and they aren't necessarily satisfied by the same things. There is a good reason why popular restaurants have many entrees on the menu.
- *The information needed to make effective software architecture decisions is often left unstated.* All systems are deployed in environments that place significant obstacles in their path. Overcoming these obstacles is the responsibility of every system, and succeeding in spite of them is the mark of an effective system. However, unless developers have an extremely deep understanding of the problem

Table 1. Three vital traits of an intentional system

Trait	Mechanism	Example
Provide useful features	Provide a significant new capability	Given Imaging Inc. developed a 11x26mm capsule that encases a digital camera that is capable of passing through and taking images of parts of a patient's GI tract.
	Improve the quality of existing capabilities	Intel's Pentium 4 CPU uses a 90mm design rule (reduced from 130mm) and is able to perform 13 billion instructions per second.
Overcome obstacles	Address limiting factors	Many mutual funds and privately managed portfolios are obligated to meet investment constraints. Pre-trade compliance systems analyze proposed trades to verify that the portfolio remains in compliance.
	Identify and mitigate risks	A wind shear detection system in a commercial airplane detects the presence of microbursts of wind that could cause an airplane that is in the process of landing to crash.
Cope with change	Exploit opportunities	eBay recognized that the rapidly growing population of consumers with Internet access created an opportunity to provide an electronic auction capability.
	Adapt quickly to new conditions	Eastman Kodak recognized the technology shift that enabled digital photography and achieved market penetration in this segment to offset declines in film sales. Polaroid was not as successful in doing so.

domain, they haven't accumulated the acumen to make good judgments. At the same time, influential users and system sponsors usually do have this experience, but often lack the technology or systems expertise to know when it is needed.

Agile Development

Agile methods, like extreme programming (XP) and Scrum, take a slightly different approach. These methods emphasize some useful changes, such as close collaboration between stakeholders and developers, and very short project iterations to get continual feedback. The theory is that continuous interaction between stakeholders and developers is a more reliable mechanism for project navigation, than a big up-front investment in written requirements.

In addition, agile methods tend to favor more organic, reactive approaches (refactoring) to those with more prescriptive guidance (architecture). Proponents of agile methods speak of allowing the architecture of a system to evolve. In some situations, this approach can be effective. One example is when user needs or competitive conditions change rapidly. However, there are many cases where this approach can be risky. One in particular is when a product must be developed to run in many different environments and/or satisfy stakeholders with different needs and priorities.

The main issue with waterfall, spiral, and agile approaches is that software development often proceeds without some very critical information, and without the tools needed to gather it. A seaworthy boat, a working radio, and a complete set of sails are all necessary, but not necessarily sufficient. An experienced sailor wouldn't think of leaving port without a good set of nautical maps, a long-range weather forecast, and a reliable way of tracking the boat's location.

We'll discuss two processes here: value modeling and architecture strategy. It will show how effective use of these techniques will capture essential information about the problem domain that enables users and developers to make effective trade-offs, permit significant obstacles to success to be identified and prioritized, and enable the architecture strategy to be expressed in a clear, concise way that can be understood by all stakeholders.

Purposeful systems are developed to create value for their stakeholders. In most cases, this value is perceived to be beneficial because these stakeholders play important roles in other systems. In turn, these other systems exist to create value for their stakeholders. This recursive quality of systems is one key in the analysis and understanding of value flows (we'll discuss this point in more depth shortly).

Table 1 lists three vital traits of an intentional system. Two mechanisms to achieve each trait are described, and a real-world example is provided for each mechanism. These three traits are at the heart of a value model. To identify and work with them more easily, we need to reduce each one down to an elemental form:

- Value expectation expresses a need for a particular feature, including what is provided (capabilities); how well they are provided (quality attributes); and how beneficial are various levels of quality (utility function). For example, one driving a car might have a value expectation for how quickly and safely the vehicle can stop from a speed of 60 miles per hour.
- Opposing force represents some natural or imposed force in the environment where a system is deployed that makes satisfying a value expectation well more difficult. For example, how effectively a car can stop from 60 miles per hour depends on the type of surface (pavement or gravel), slope (up or downhill), conditions (dry, wet, or ice), and the weight of the vehicle.
- Change catalyst represents some force or event in the environment that causes value expectations to shift or limiting factors to have a different impact. For example, decreases in memory chip costs and increases in storage density became a catalyst for digital photography.

For the remainder of this discussion we'll refer to opposing forces and change catalysts as limiting factors, and we'll refer to all three collectively as value drivers.

Architecture Challenges in Portfolio Compliance Systems

Compliance rules might specify upper and lower limits on the percent of the portfolio's assets that can be invested in particular categories, such as security type, industry sector, or geographical region. Allocation percentages in a portfolio can vary based on price changes, trades, and corporate actions. If a portfolio fails to comply with its rules and loses money, it may be required to indemnify its investors.

In situations where a portfolio is well within its compliance limits, there is a low risk that any particular trade will cause a violation. However, if the market is moving rapidly or trade volume is heavy, a portfolio that is close to one or more limits has a much higher risk of becoming out of compliance.

One major challenge is that rapidly moving markets or periods of heavy trade volume are exactly the scenario where traders must be able to respond quickly, to get the best prices. Yet, this scenario can be the same situation where there are many pending trades and price-change events to evaluate, making the compliance verification more complex.

In this case, portfolio size, trade volume, and market volatility all combine to create a conflict between the need for compliance verification (a risk-mitigation technique) and the need for efficient trading (which impacts portfolio ROI). To be effective, the architectures of the portfolio management organization and its information systems must find a way to balance the trade-off between compliance risk and timely trades. •

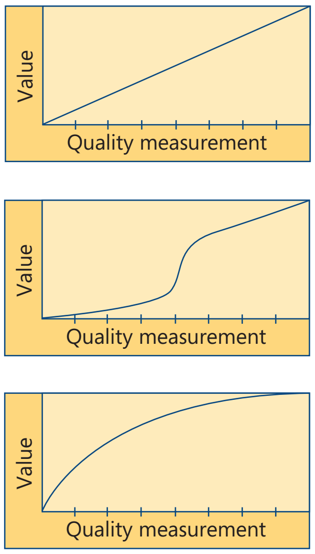
Not So Simple

If a system is to be effective at satisfying the value models of its stakeholders, it needs to be able to identify and analyze them. Traditional approaches, like use-case scenarios or business/marketing requirements, start by focusing on the types of actors with which the system interacts. This approach has several major limitations: it focuses more on what things the actors do, and less on why they do them; it tends to stereotype actors into categories, where all individuals of a type are essentially the same (for example, traders, portfolio managers, and system administrators); it tends to ignore differences in limiting factors (for example: Is an equity trader in New York the same as one in London? Is trading at market open the same as trading during the day?); and it is based on binary outcomes: the requirement is met or it isn't. The use case completes successfully or it doesn't.

There is a very logical, practical reason why this approach is popular. It uses sequential and classification-based reasoning, it is easy to teach

Figure 1. Utility curves

Value level	Miles per gallon	Perceived utility		
		Linear	S-Curve	Parabola
Worst	10 mpg	0	0	0
Adequate	20 mpg	25	10	60
Satisfactory	30 mpg	50	50	80
Preferable	40 mpg	75	85	90
Best	50 mpg	100	100	100



and explain, and it can produce a set of objectives that are easy to verify. Of course, if simplicity were the only goal that counted, we'd all still be walking or riding horses to get from one place to another.

In his book *Competitive Advantage: Creating and Sustaining Superior Performance*, Michael Porter discusses the concept of value chains in the context of corporate strategic planning (see Resources):

"Although value activities are the building blocks of competitive advantage, the value chain is not a collection of independent activities, but a system of interdependent activities. Linkages are relationships between the way that one value activity is performed, and the cost or performance of another.

"Linkages exist not only within a firm's value chain (horizontal linkages), but between a firm's value chain and the value chains of suppliers and channels (vertical linkages). The way that supplier or channel activities are performed affects the cost or performance of a firm's activities (and vice versa)."

If one thinks of a firm (or a supply chain) as a system, and each major value activity (procurement, receiving, manufacturing, and so on) as a subsystem, then we can generalize the notion of value chains and linkages: each entity (value activity) has its own value model to represent its value expectations and limiting factors, each linkage describes how the value model of one entity dovetails with the value model of the entity with which it is linked, and each linkage between two entities in the same system is what Porter refers to as a horizontal linkage. Each linkage between entities in different systems is a vertical linkage.

Porter also refers to the concept of differentiation, where two entities performing the same set of value activities behave differently. A simple example might be a taxi versus a municipal bus. While both provide ground transportation for a fee, these two contexts have different features. The bus is relatively inexpensive and follows a predetermined route and schedule. The taxi is available on demand (except for when you really need one), operates point-to-point, is more expensive, and holds a limited number of passengers. When it is raining, the extra cost of a taxi might not matter as much.

Question of Balance

For the rest of this discussion we will use the term *value cluster* to refer to an abstract entity that performs a general type of value activity. *Value context* will be used to refer to a specialized form of a value cluster that has significant differences in value expectations, opposing forces, or change catalysts from other contexts in the same cluster.

Both value clusters and value contexts have their own value models. The value model of a cluster represents the common aspects of all contexts that specialize that cluster. Each value context specializes the value model of its cluster. The set of value models for all contexts in a cluster provide important insights into the differences between what each one expects, and how its environment affects it.

Why is this point important? A system's architecture must perform a delicate balancing act involving its value drivers. This can be tricky in a single-context system, where all deployment scenarios have equivalent value expectations and limiting factors. Tasters and AA batteries

"THE MAIN ISSUE WITH WATERFALL, SPIRAL, AND AGILE APPROACHES IS THAT SOFTWARE DEVELOPMENT OFTEN PROCEEDS WITHOUT SOME VERY CRITICAL INFORMATION, AND WITHOUT THE TOOLS NEEDED TO GATHER IT"

are good examples of single-context systems. So are simple text editors, file difference analyzers, and many other PC desktop utilities. In a single context system, it is still possible to have interdependencies and conflicts among combinations of value expectations and limiting factors.

However, it gets more challenging. Most complex systems have multiple contexts. In other words, as you consider different deployment environments, they have significant variation in value expectations, opposing forces, and change catalysts. As either the number of contexts increases, or their degree of compatibility decreases, it becomes much more difficult to satisfy all of them with a single architecture. While there are several techniques for dealing with this situation, the first step is to recognize when you face them.

Many systems have only a few contexts. These occur most often with systems that are deployed for internal use inside an organization. Different deployment environments can have different limiting factors. For example, a system for dispatching airline baggage handlers is affected by weather extremes, or an international system is affected by local regulations. Other times, deployment environments have different value expectations. This is especially true when there are international or cultural differences. Nurses who operate haemodialysis machines for patients with chronic kidney failure in a government-sponsored hospital in Europe will have different wants and priorities than nurses who perform the same task in a small, private walk-in clinic in the U.S. (where private insurance providers pay for treatments).

Many other systems have a large number of contexts. These occur most frequently with technology-centric products that are developed for sale or lease to a wide array of customers. The same conditions that cause variation in slight context systems occurs in spades because the number of deployment contexts can be thousands or millions of times larger, the organizations (or systems) in which the stakeholders participate can have very different sets of value expectations, and the catalysts that trigger significant change in each deployment environment are likely to be very different.

In summary, a value model captures the drivers that determine how satisfied a particular market segment is, and how difficult it will be to satisfy them.

Utility Curves

Previously, this discussion made reference to an important concept called a utility curve. Very simply, a *utility curve* is a mapping from one scale of measurement to a second. The first scale represents a result variable that can be quantified. The second scale is the level of value (satisfaction, utility) that is generated. The most common example of a utility curve is one used to map test scores into letter grades for a high school or college exam. As you will see, a good grasp of utility curves is absolutely essential to making effective trade-off decisions.

Figure 1 illustrates a simple example. The first scale represents the EPA combined city and highway fuel economy for a vehicle. The second scale represents five qualitative values: worst, adequate, satisfactory, preferable, and best. Worst is the minimum passable requirement, and little or no value is lost with results below this level. Adequate represents a below-average outcome—disappointing, but acceptable. Satisfactory is the expected outcome—no better, no worse. Preferable represents an above-average outcome that is satisfying and pleasing, but not far above the range of ordinary. Best is the best expected outcome, and little or no value is gained with results that exceed it.

The example in Figure 1 shows three distinct utility curves. There are many other possible shapes; these represent three common ones. The first curve is linear, the second has an s-curve shape, and the third is a parabola. All three have the exact same worst and best values. What is interesting to note is the intermediate values. An increase from 10 to 20 miles per gallon yields 10 percent of the available value for the s-curve, but 60 percent for the parabola.

In a single-context system the use of utility curves to analyze architecture strategies is straightforward. The decision analysis method described by Charles Kepner and Benjamin Tregoe in their book can be used for this purpose (see Resources). Each alternative is evaluated against each value expectation. Utility curves are used to map the value of the quantitative measure achieved by each alternative to its corresponding value. Then the value levels are weighted by the priority of the expectation, and totaled. More preferable alternatives have higher totals.

The most challenging aspect of this method is choosing an appropriate mechanism to evaluate each alternative against each want goal. The best scenario is when the mechanism provides an objective mea-

Figure 2. Architecture strategy formulation

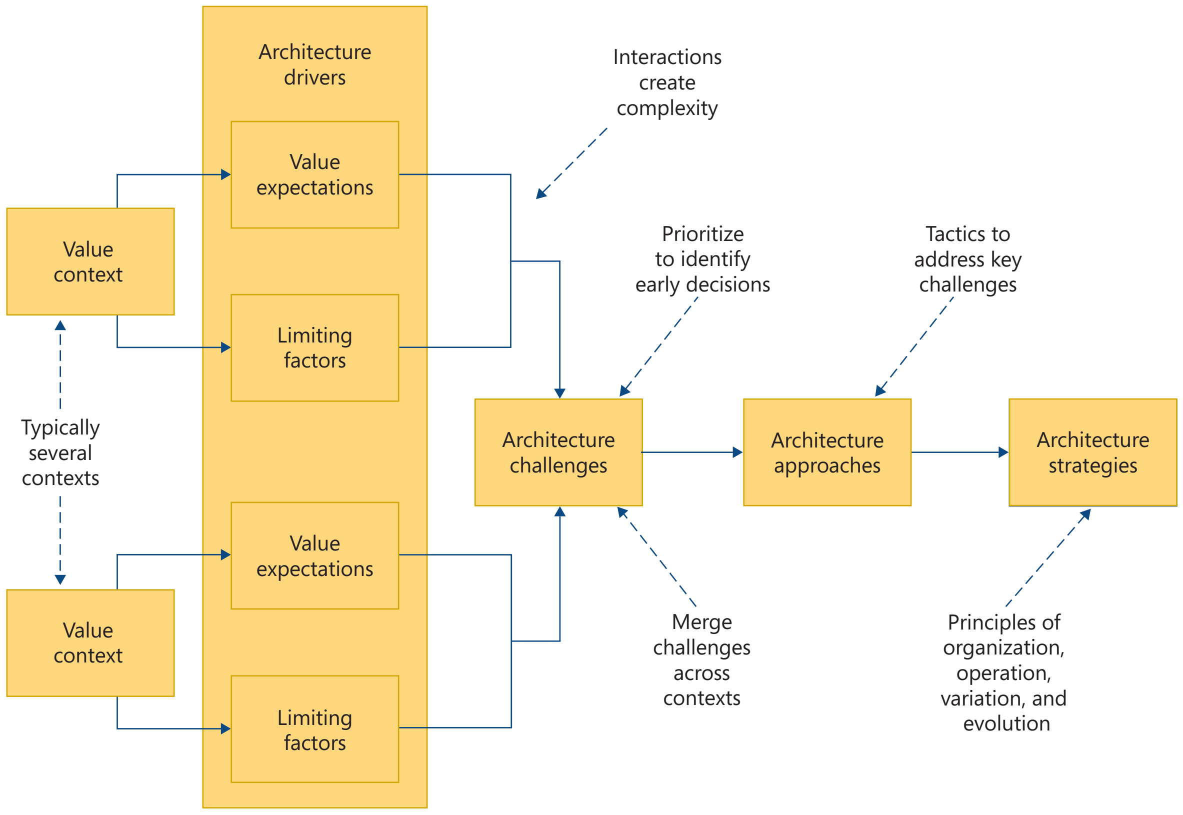
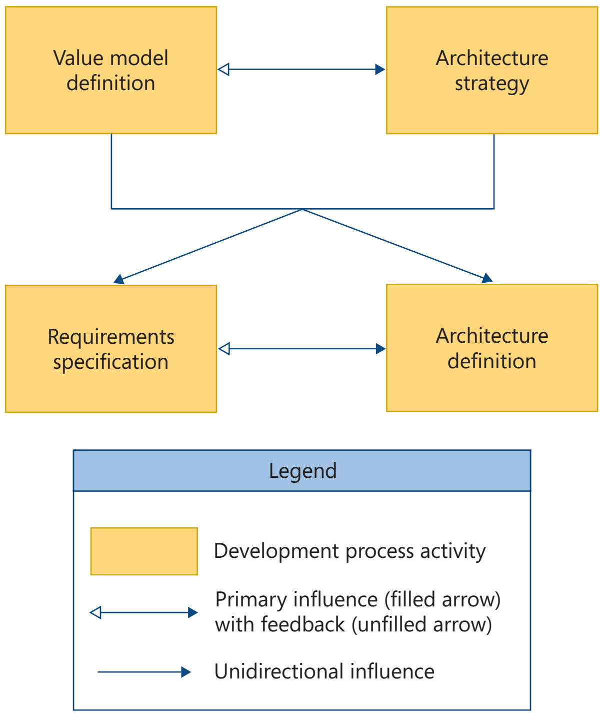

Architecture drivers
Value expectations
Limiting factors
Value expectations
Limiting factors
Value context
Value context
Typically several contexts
Interactions create complexity
Prioritize to identify early decisions
Tactics to address key challenges
Architecture challenges
Architecture approaches
Architecture strategies
Merge challenges across contexts
Principles of organization, operation, variation, and evolution

Figure 3. Value-driven architecture with traditional methods


Value model definition
Architecture strategy
Requirements specification
Architecture definition
Legend
Development process activity
Primary influence (filled arrow) with feedback (unfilled arrow)
Unidirectional influence

surement (such as measuring miles per gallon or horsepower for an automobile engine). In some cases, the mechanism might be subjective. The cost of coming up with an appropriate objective measurement must be balanced against the extra accuracy and objectivity provided. In some situations, an initial assessment can be done with subjective assessments. If the results are close, then objective measurements can be made to choose among the best alternatives.

Architecture Challenges

An *architecture challenge* is a situation where one or more limiting factors make it more difficult to satisfy one or more value expectations. Simply put, an architecture challenge is an obstacle or barrier that the system must overcome to provide value. This is a key point. Obstacles and value expectations are like yin and yang. If obstacles are not present, then value drops because the outcome is easy and anyone can do it. Bottled water is the one noteworthy exception to this rule. Within any context, identification of architecture challenges involves assessing which limiting factors impact one or more value expectations. If impacts are observed, do they make fulfilling the value expectation(s) easier (positive impact) or harder (negative impact)? And how hard or easy does each impact make things? A simple low, medium, or high scale usually is sufficient here.

The sidebar, "Architecture Challenges in Portfolio Compliance Systems," describes some architecture challenges that occur in that kind of compliance system. A more in-depth discussion of architecture challenges and a case study can be found in a whitepaper by the author (see Resources).

Architecture challenges must be considered within their own context. While it might be possible to average utility curves across contexts, the same cannot be done with the impact of limiting factors on value expectations. For example, suppose a Web server supplies pages to users in two contexts. One context accesses static information, such as reference documents. They want response times between 1–3 seconds. The other context accesses very dynamic information, like box scores of in-progress sporting events. They are satisfied with response times in the range of 3–6 seconds.

Both contexts are subject to CPU, memory, disk, and network limitations. However, as request volumes increase by a factor of 10 or 100, these two contexts are likely to run into very different scalability obstacles. In the dynamic content case, synchronization of updates and accesses becomes a limiting factor under heavy load. For the static content, heavy load can be overcome by caching frequently read pages.

There is one final point that should be mentioned about architecture challenges and multiple-context systems. In many cases, it will seem that a single system is capable of supporting many different contexts. However, the architecture contexts that arise from each context are a very good tool for evaluating how compatible these contexts are with each other. When incompatible contexts are addressed by the same architecture, the result is never that both are satisfied. Either one suffers at the expense of the other, or both are compromised. One example of this situation is a semiconductor tool that attempted to support production and research contexts with a single architecture. Given the very different sets of value expectations (reliability versus flexibility), opposing forces (fab versus lab), and change catalysts (production runs versus experiments), it was unlikely that this marriage could be saved.

Architecture Strategy

As discussed earlier, formulating a system's architecture strategy starts with recognizing the appropriate value contexts and prioritizing them, defining utility curves for and prioritizing value expectations in each context, identifying and analyzing opposing forces and change catalysts in each context, and detecting where limiting factors make it hard to fulfill value expectations.

Figure 2 illustrates this process. The previous list of activities brings us into the architecture challenges box in the middle of the diagram.

"IF SIMPLICITY WERE THE ONLY GOAL THAT COUNTED, WE'D ALL STILL BE WALKING OR RIDING HORSES TO GET FROM ONE PLACE TO ANOTHER"

At this point, we are working with a list of architecture challenges that have been gathered from all of the contexts. Each of these challenges represents the impact of one or more limiting factors on one or more value expectations. As the diagram shows, before we start addressing each challenge, we need to prioritize them. The observations we'll discuss here explain why the earlier a decision is made, the more things it is likely to constrain, and the later a decision is made, the fewer alternatives there are available.

As a result, it only makes sense to reserve the earliest architecture decisions to be the ones that yield the most value. There are several criteria that can be used for prioritizing architecture challenges. We recommend a balance among:

- Importance – How high is the priority of value expectations that are impacted by the challenge? If these value expectations are specific to a few contexts, then what is the relative priority of these contexts?

- Magnitude – How large of an impact on the value expectations was caused by the limiting factors?
- Consequence – How many realistic options do there appear to be? Do these options have significant differences in difficulty or effectiveness?
- Isolation – How isolated is the impact of the most realistic options? The more widespread the impact, the more weight this factor has.

Once the architecture challenges are prioritized, approaches are formulated for those that are highest priority. While techniques such as architecture styles and patterns can help, this is an area where deep experience with the problem and solution domains is invaluable (see Resources for links to two books on these topics). Effective approaches to significant challenges are the result of skill, insight, effort, and painstaking work. This statement is true, regardless of whether the problem is surgery, executive management, or software architecture.

As each challenge is addressed, its approach will constrain the solutions to other challenges, and sometimes create new ones. If the architecture challenge priorities are correct, then most of the downstream constraints will be appropriate. However, in some cases, the approach to a high-priority challenge might negatively impact several slightly lower-priority challenges. The combined priority of the impacted challenges might outweigh the higher-priority challenge. In this case, it is advisable to back up and formulate a different approach to the original challenge.

Set Sail

Once approaches have been formulated to the set of high-priority challenges, the architecture strategy can be expressed. The architect analyzes the set of approaches, and factors out a set of guiding principles in these areas:

- Organization – How is the system organized into subsystems and components? What is the composition and responsibilities of each? How can the system be deployed over a network? What types of users and external systems are there? Where are they located and how do they connect?
- Operation – How do components interact? In which cases is communication synchronous? In which cases are they asynchronous? How are the actions of components coordinated? When is it acceptable to configure a component or run diagnostics on it? How are error conditions detected, diagnosed, and corrected?
- Variability – Which major features of the system are permitted to vary from one deployment environment to another? Which options are supported for each feature, and when can the choice be made (for example, compile, link, installation, startup, or at runtime)? What dependencies are there between variation points?
- Evolution – How is the system designed to support change while retaining its stability? Which specific types of significant change have been anticipated, and what are the preferred ways to address them.

In summary, the architecture strategy is the rudder and keel of a sailboat, providing direction and stability. It is expected to be a brief, high-level statement of direction that must be understandable by all stakeholders, and should be relatively stable over the lifetime of the system.

Figure 4. Integration of value-driven architecture with agile methods

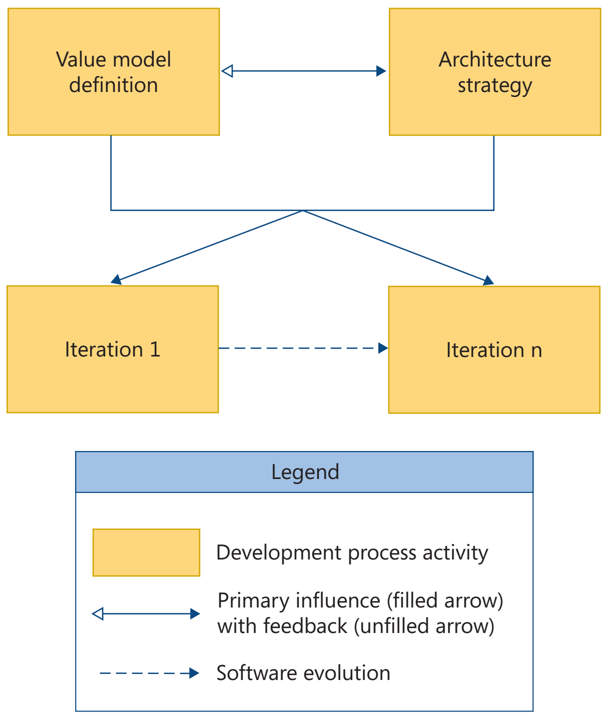

Value model definition
Architecture strategy
Iteration 1
Iteration n
Legend
Development process activity
Primary influence (filled arrow) with feedback (unfilled arrow)
Software evolution

Figure 3 shows how value models and architecture strategy relate to the waterfall and spiral methods. Value models and architecture strategy operate at both an earlier point and a higher level than these methods. When value models are studied and architecture strategies are formulated, they provide a great foundation for specifying requirements and defining a more detailed architecture. The value model drives the requirements and influences the architecture definition by providing information for making trade-offs. The architecture strategy drives the more detailed architecture definition and provides a set of derived requirements that are needed to overcome known obstacles.

An appropriate analogy is to view architecture strategy as strategic planning, and value models as market analysis. In this light, requirements become corporate objectives and policies. Architecture definition is the business organization and operational plan, and use cases are the equivalent of business processes.

Few companies establish corporate objectives, organizational structure, operating plans, and business processes without first having a clear idea of their mission, markets, competitors, resources, and strategy. Even fewer effective ones do this.

Figure 4 shows how value models and architecture strategy relate to agile methods. Both XP and Scrum make allowances for an architecture definition. Scrum does this explicitly, expecting the architecture to be defined in the first 4–5 week iteration. XP does this implicitly. One of the 12 core principles of XP is called system metaphor. This principle is not used as frequently or is as well understood as its more famous siblings: small releases, pair programming, and test-driven development.

In the early days of XP, the team that worked on the large, complex Chrysler Payroll System needed a good way to describe workflow management to the Chrysler developers. Somebody got the idea of drawing an analogy between payroll workflow and an automotive assembly line. The metaphor clicked, and the Chrysler developers got the picture.

Tell a Story

The XP Web site defines system metaphor as what XP uses instead of a formal architecture. A simple shared story of how the system works, a metaphor, typically involves a handful of classes and patterns that shape the core flow of the system being built.

What XP refers to as a "formal architecture" is more like what was referred to previously in this discussion as an architecture definition. An architecture strategy plays the same role as a system metaphor, without being a metaphor. This definition is a significant advantage, since really effective metaphors (such as the one used in Chrysler) can be hard to come by. By contrast, clear, concise core principles are easy to state and easy to understand. A person doesn't need to go out and watch the movie Hidalgo to understand what is meant by "life, liberty, and the pursuit of happiness."

In summary, the value model helps us to understand and communicate important information about sources of value. Some of the important issues it addresses are how value flows, why similarities and differences occur in value expectations and external factors, and what subset of that value our system seeks to satisfy. It is the architect's job to satisfy these value expectations by resolving forces that influence the system in general—forces that are specific to certain contexts and forces that are expected to change over time. In this respect, architecture is similar to flying a jet airplane: the pilot must transport passengers safely to a known destination, while balancing the laws of aerodynamics, the capabilities of the plane, and current and future weather conditions.

The link between value models and software architecture is clear and logical, and can be expressed by these nine points:

1. Software-intensive products and systems exist to provide value.
2. Value is a scalar quantity that incorporates perceptions of marginal utility and relative importance across many distinct goals. Trade-offs between goals are an extremely important consideration.
3. Value exists at multiple levels, some of which contain the target system as a value provider. The value models for these scopes contain the primary drivers of the software architecture.
4. Value models that are above these levels in the hierarchy can cause the value models of their children to change, which is important input in formulating the principles of evolution for the system.
5. For each cluster, value models are homogeneous. Value contexts, exposed to different environmental conditions, have different expectations of value.
6. The development sponsor for the system has different priorities for trying to satisfy various value contexts.
7. Architecture challenges result from the impact of environmental factors on value expectations within a context.
8. Architecture approaches seek to maximize value by addressing the highest-priority architecture challenges first.
9. Architecture strategies are synthesized from the highest-priority architecture approaches by factoring out common rules, policies, and principles of organization, operation, variation, and evolution.

The main contributions of this approach are:

- The sources of value in the system are modeled as first-class concepts. Value expectations associate a small number of capabilities with quality attributes, utility curves, and external factors. Value expectations are held by value realms and contexts; realms capture the common aspects of value expectations, while contexts capture the important multiple variability within a realm.
- Traceability of architectural reasoning is also a first-class entity. Value expectations link to architecture challenges, which link to architecture approaches, which link to architecture strategies. Stakeholders now can see the thought process that went behind the solution.
- A very useful side effect of this traceability is an increased ability to review software architectures. Because the reasoning behind the decisions is made explicit, it becomes easier for other stakeholders (project sponsors, domain experts, technology experts, end users) to identify aspects that might be missing or incorrect. •

RESOURCES

Competitive Advantage: Creating and Sustaining Superior Performance, Michael E. Porter (Free Press 1998)

Design Patterns: Elements of Reusable Object-Oriented Software, Erich Gamma et al. (Addison Wesley 1995)

http://c2.com/cgi/wiki?ExtremeProgrammingCorePractices
Extreme Programming Core Practices

"Making Architecture Decisions, an Economic Approach,"
Rick Kazman et al. Technical Report (Carnegie Mellon University, Software Engineering Institute 2002)

The New Rational Manager, Charles H. Kepner and Benjamin Tregoe (Kepner-Tregoe Inc. 1997)

Pattern-Oriented Software Architecture, Volume 1: A System of Patterns, Frank Buschmann et al. (John Wiley and Sons 1996)

Software Factories: Assembling Applications with Patterns, Models, Frameworks, and Tools, Jack Greenfield et al. (Wiley 2004)

www.foliage.com/whitepapers/index.shtml
"Using Architecture Challenges to Formulate Software Architecture," Charlie Alfred (Foliage Software Systems Inc. 2003)

About the Author

Charles Alfred is technical director at Foliage Software Systems, which delivers competitive advantage through technology strategy, software architecture, and custom software development. Since being founded in 1991, Foliage has completed more than 175 projects for clients in financial services, semiconductors, health care, aviation, and e-business.

Enterprise Architect Summit 2005

November 6 - 8
Barcelona, Spain

Register by the Early Bird Deadline of October 5 and
Save €200*

*Savings Rate is subject to Euro/USD exchange rate fluctuations

Build a Solid Foundation for the Agile Enterprise

After three successful North American events, Enterprise Architect Summit sets its sights on Europe this autumn, premiering with three days of keynotes, workshops and special activities in cosmopolitan Barcelona.

Join us for in-depth sessions highlighting IT strategies and best practices from the real world—topics to help equip your organization to respond to emerging IT challenges and opportunities.

Register Now! Call: +1 650.378.7100

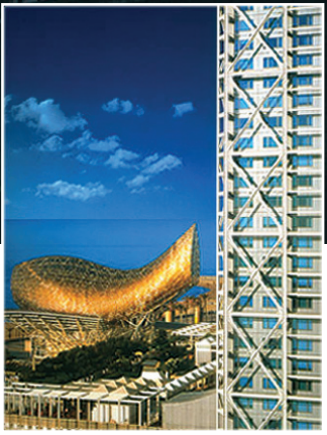
Or visit: www.ftponline.com/conferences/eas/barcelona

Visiting Barcelona

Few European cities offer the wide diversity of cultural experience that you'll find in Barcelona. Couple that with the luxury of 4.2 km of beach only a short walk from the city centre and glorious autumn weather, and you have all the makings of a great destination.

Register Now! Call: +1 650.378.7100

Or visit: www.ftponline.com/conferences/eas/barcelona



Take Advantage of Great Rates and Maximum Convenience

Stay On-Site at the Ritz-Carlton Hotel Arts, the Official Hotel for Enterprise Architect Summit

Be in the heart of it all! Make your time at EAS even better by staying at the Hotel Arts on the shores of the Mediterranean, where all conference sessions and activities will be held. The striking contemporary hotel enjoys a spectacular beachfront location near Barcelona's finest shops and restaurants. The graceful silhouette of Hotel Arts Barcelona offers guests spectacular views over the city and sea.

As an EAS attendee, you are eligible for a special room rate of €230 (excl. VAT) for single/double occupancy. Make your reservation early because the discounted rate is only in effect until October 3, 2005 and rooms are subject to availability. Visit www.ftponline.com/conferences/eas/barcelona for details.

Microsoft®
Founding & Platinum Partner

FTP FAWCETTE TECHNICAL PUBLICATIONS
1990-2005 15TH ANNIVERSARY

THE
ARCHITECTURE
JOURNAL

Microsoft®

098-103796