Principles of Service-Oriented Computing

- Use Cases
  - Intraenterprise Interoperation
  - Interenterprise Interoperation
  - Application Configuration
  - Dynamic Selection
  - Software Fault Tolerance
  - Grid
  - Utility Computing
  - Software Development

Intraenterprise Interoperation :

Intraenterprise interoperation is about achieving the interoperation of applications within an enterprise. This set of problems is also known—mistakenly, in the view of the authors—as enterprise integration. The classical problem here is to make different software components work well together. These components would often have begun their existence as independent, self-contained systems.

Considering only the interoperation aspects of the above problem, we can easily see that several challenges must be overcome. First, there must be connectivity among the applications, nowadays readily ensured through the ubiquity of IP networks. Higher-level protocols such as TCP and HTTP facilitate this further.

Interenterprise Interoperation:

Interenterprise interoperation is fast becoming an important application setting for IT. Previously, enterprises interoperated through *ad hoc* means that required substantial human intervention. Or, they used rigid standards such as Electronic Data Interchange (EDI), which led to systems that were difficult to maintain. Recently, there has been growing interest in supplychain management and intelligent manufacturing, leading up to cross-enterprise processes in general. The idea is that businesses must work together anyway. If they can streamline their interactions through technology, they can improve their responses to information, reduce overhead, and exploit emerging opportunities.

Service-oriented computing provides the same benefits as for intraenterprise interoperation above. In addition, it provides the ability for the interacting parties to choreograph their behaviors so that each may apply its local policies autonomously and yet achieve effective and coherent cross-enterprise processes.

Application Configuration:

Service-oriented computing enables the customization of new applications by providing a Web service interface that eliminates messaging problems and by providing a semantic basis to customize the functioning of the application.

Imagine that the hospital purchases an anesthesia information management system (AIMS) to complement its existing systems. An AIMS would enable anesthesiologists to better manage anesthetic procedures on patients in surgery and to better monitor, record, and report key actions, such as the turning on or turning off of various gases and drips. Such information can help establish compliance with government regulations, ensure that certain clinical guidelines are met, and support studies of patient outcomes.

Dynamic Selection:

Imagine that a hospital wishes to purchase supplies such as catheters. To carry out such purchases efficiently requires that the hospital be able to interoperate with the catheter vendor—a case of interenterprise interoperation. Now suppose that the hospital would like to purchase catheters from whichever vendor offers it the best terms.

In other words, the business partner—the other enterprise with which to interoperate—would be chosen on the fly. Such dynamic selection is becoming increasingly common as the possible gains of such flexibility are recognized. If business partners can be selected flexibly, then they can be selected to optimize any kind of quality-of-service criteria, such as performance, availability, reliability, and trustworthiness.

Service-oriented computing enables dynamic selection of business partners based on quality-of-service criteria that each party can customize for itself.

Software Fault Tolerance :

Suppose a hospital is carrying out a business transaction with a partner and encounters an error. It would be great if the interaction could be rewired to an alternative business partner dynamically in a manner that is transparent to the overall process. To the extent that the state of the interaction is lost, some means of recovery would be needed to restore a consistent state and resume the computation with new business partners.

Service-oriented computing provides support for dynamic selection of partners as well as abstractions through which the state of a business transaction can be captured and flexibly manipulated; in this way, dynamic selection is exploited to yield application-level fault tolerance.

Grid :

Grid computing refers to distributed computing where several resources are made available over a network, and are combined into large applications on demand. Grid computing is a form of *metacomputing* and arose as the successor of previous approaches for large-scale scientific computing. Building complex applications over Grid architectures has been difficult, which has led to an interest in the more modular kinds of interfaces based on services. Accordingly, Grid services have been proposed in analogy with Web services. *Service-oriented computing enables the efficient usage of Grid resources.*

Utility Computing:

Following up on Grid-like environments, there has been a recent expansion of *utility computing*, where computing resources are modeled as a utility analogous to electric power or telecommunications. The idea is that enterprises would concentrate on their core business and out-source their computing infrastructure to a specialist company. Leading companies such as IBM and HP have made utility computing offerings; the one from IBM is called *autonomic computing*, although that term is starting to be used generically as a technical area for fault-tolerant computing. Utility computing presupposes that diverse computational resources can be brought together on demand and that computations can be realized on physical resources based on demand and service load. In other words, service instances would be created on the fly and automatically bound to dynamically configure applications.

Service-oriented computing facilitates utility computing, especially where redundant services can be used to achieve fault tolerance.

Software Development:

Software development remains a challenging intellectual endeavor. Improvements are achieved through the use of superior abstractions. Services offer programming abstractions where different software modules can be developed through cleaner interfaces than before. When the full complement of semantic representations are employed, the resulting modules are not only more easily customizable than otherwise, but the following holds:

Service-oriented computing provides a semantically rich and flexible computational model, for which it is easier to produce software.