

Architecture Paradigms and Their Influences and Impacts on Component-Based Software Systems

Guijun Wang, Casey K. Fung
Mathematics and Computing Technology
Boeing Phantom Works
Seattle, WA 98124 USA

Email: guijun.wang@boeing.com; casey.k.fung@pss.boeing.com

Abstract

Object-Oriented Architecture (OOA), Component-Based Architecture (CBA), and Service-Based Architecture (SBA) represent three technical architecture paradigms in current software systems. Object, Component, and Service are three key concepts in distributed software systems. From implementation point of view, a service is implemented by one or more components, which in turn are often implemented in object-oriented programming languages like C++ and Java. Distributed component-based software systems can be structured in any of the architecture paradigms, some have more advantages than others depending on business requirements. Understanding the characteristics, features, benefits, and concerns of the architecture paradigms is crucial to the successful design, implementation and operation of a distributed system.

In this paper, we describe the characteristics of the three architecture paradigms and the business drivers for their applications. The parallel evolution of architecture paradigms and software development methodologies is discussed in the context of practical business needs. Component-based software developers for distributed systems should decide on the architecture paradigms based on business requirements. The evolution of architecture paradigms and the selection of architecture paradigms have profound influences and impacts on component-based distributed systems, in the way components are designed and in the way component interactions are implemented. We discuss these influences and impacts with the goal of deriving some practical principles and strategies to best deal with them in software engineering practices.

1. Introduction

Architecture design plays a prominent role in software engineering processes. Regardless of the variations in

software engineering processes, software architecture provides the skeleton and constraints for software implementation. Software architecture is defined as “the fundamental organization of a system embodied in its components, their relationships to each other, and to the environment, and the principles guiding its design and evolution” [17]. The concept of ‘component’ is loosely defined however, and it allows the architecture definition to accommodate components of all types and various granularities. Software engineering is an iterative process that comprises multiple stages, including modeling, design, implementation, deployment, and maintenance. The “component” concept has been utilized in all stages to denote parts of the system or a subsystem. Because of the variation in component conceptualization, there are several architecture paradigms according to how the concept is perceived in software architecture. The most well recognized architecture paradigms are Object-Oriented Architecture (OOA), Component-Based Architecture (CBA), and Service-Based Architecture (SBA).

The evolution of these architecture paradigms has deep and subtle influences and impacts on software systems, in the way components are designed and structured and in the way component interactions are implemented. The evolution is driven by business integration needs both in terms of enterprise application integration and distributed systems integration. Technical enablers such as component technologies, web technologies, and web services standards have helped speed up the evolution.

Business integration needs include common business data, flexible business processes, seamless intra-enterprise and inter-enterprise collaborations, and increasing connectivity and interoperability between distributed systems. Web technologies such as HTTP servers, XML-based data representation, and standard business process languages (e.g., ebXML) enable the enterprise to offer services to clients anywhere and anytime. Component technologies such as component-based application servers and integration servers have simplified the development and deployment of business components and reduced

enterprise integration time. The joint force of business integration needs, web technologies, and component technologies underpin the architecture evolution of enterprise systems.

Component-Based Software Systems (CBSS) are based on the principles of component-based software engineering (CBSE) [20] [19] [18] [21]. The component concept is more rigorously defined in CBSE. While there are a number of component definitions, Szyperski [20] offers a useful general definition - a software component is a unit of composition with contractually specified interfaces and explicit context dependencies. In this paper we adopt this definition and more specifically define a component as an artifact designed and developed based on a component model such as CORBA Component Model [4], J2EE (JavaBeans and Enterprise JavaBeans) [3], and .Net [7].

From technology and engineering viewpoints [16], components and their interactions in CBSS are implemented in programming languages and on distributed middleware platforms. Given the current dominance of object-oriented programming languages like C++ and Java, it is not always a straightforward design decision on which of the three architecture paradigms to structure a distributed system. The selection of any of the paradigms will in turn constrain the properties (e.g., performance, extensibility, and flexibility) of the system and the way components and their interactions are implemented in the system. Questions then arise on the characteristics of the three architecture paradigms and their differences in terms of features and benefits. Practical principles and strategies are to be derived in order to make sound architecture design decisions.

In the remaining of the paper, we will present the characteristics of the three architecture paradigms, contrast them, and describe the business drivers for their applications. As we discuss these characteristics we will review related research work and industry trends. We will discuss the influences and impacts of architecture paradigms on component-based distributed systems and derive some practical principles and strategies to help make sound architecture design decisions in component-based software engineering practices.

2. Architecture Paradigms and Characteristics

2.1 Object-Oriented Architecture

Architecture design for distributed enterprise systems evolves along with software development methodologies and the ever changing software technologies. Object-oriented software development methodologies like the Unified Development Process [1] have been widely used

over the last decade. The Unified Modeling Language (UML) provides standardized modeling and design notions and formalisms for object-oriented software development. Systems developed based on object-oriented methodologies, modeling notions, and programming languages are often structured as a kind of Object-Oriented Architecture (OOA).

Major characteristics of OOA include

- Based on OO principles, most notably encapsulation, inheritance, and polymorphism.
- Classes of objects are the software units of modeling, design and implementation.
- Objects and their interactions are the center of concerns in architecture design.
- Separation of interfaces from implementations. An interface can be implemented by multiple classes. Interfaces also support extensions and specializations through mechanisms like inheritance.
- Transparency of distributed objects. Remote objects can be referenced and used in the same way as local objects. This does not imply that object distribution is unimportant in distributed systems. On the contrary, RM-ODP [16] devotes one of the five viewpoints, Engineering, entirely to dealing with distribution concerns in OO distributed systems.

The best example of OOA for distributed systems is the Common Object Request Broker Architecture (CORBA) [5] from OMG. CORBA defines an Object Model for specifying how distributed objects are created, used, and managed. It also defines a Reference Architecture for specifying how the interactions between distributed objects are achieved. An Interface Definition Language (IDL) is used to specify object interfaces in a language and platform neutral way. A client object obtains a reference to a server object and then invokes its methods as defined in its interfaces in the same way as a local object. An Object Request Broker (ORB) mediates the interactions between client objects and server objects in a distributed environment.

OOA can be described from multiple views. One definition of OOA views is the 4+1 views [2], namely logical, process, physical, and development plus use cases. A typical OOA description includes descriptions of these views and use cases in UML.

2.2 Component-Based Architecture

Large scale distributed systems are complex and labor intensive to develop and maintain. This gave rise to the idea that large scale distributed systems should be integrated from independently developed, well-tested, reusable software components. Component-based

software development (CBSD) takes this idea and considers components as both a modeling concept and an implementation unit in software development [21] [20]. Additional concepts like Containers in CBSD simplify the development, integration, and deployment of component-based distributed systems.

CBSD methodologies like Catalysis [19] and Kobra [18] provide processes and notations to support CBSD. Industry component models and platforms like J2EE, CORBA Component Model, and Microsoft .Net make CBSD practical. Component-based application servers and integration servers based on component models like J2EE offer standard platforms and services for deploying business components in distributed enterprise systems. The proposed UML 2.0 [6] specification from OMG also adds component-based modeling notations like components and connectors for architecture description. Systems developed based on CBSD are often structured as a kind of Component-Based Architecture (CBA).

Major characteristics of CBA include

- Based on component models.
- Components are the software units of modeling, design and implementation.

- Interfaces and interactions are the center of concerns in architecture design.
- Separation of interfaces from implementations. An interface can be implemented by multiple components. Interfaces also support extensions and specializations through mechanisms like inheritance.
- Introspection. Component models require components to support introspective operations so that their functionality and properties can be discovered and utilized at assembly time or runtime.
- Emphasize design patterns and the principle of separation of concerns for defining component roles and responsibilities.

A component model defines the creation, use, and lifecycle management of components as well as a set of supporting concepts. A component model also includes a programming model specifying how distributed components are defined, assembled, and deployed. Industry component models include CORBA Component Model (CCM), JavaBeans and Enterprise JavaBeans in Java 2 Enterprise Edition (J2EE), and .Net. Figure 1 shows a concept map for concepts in CCM.

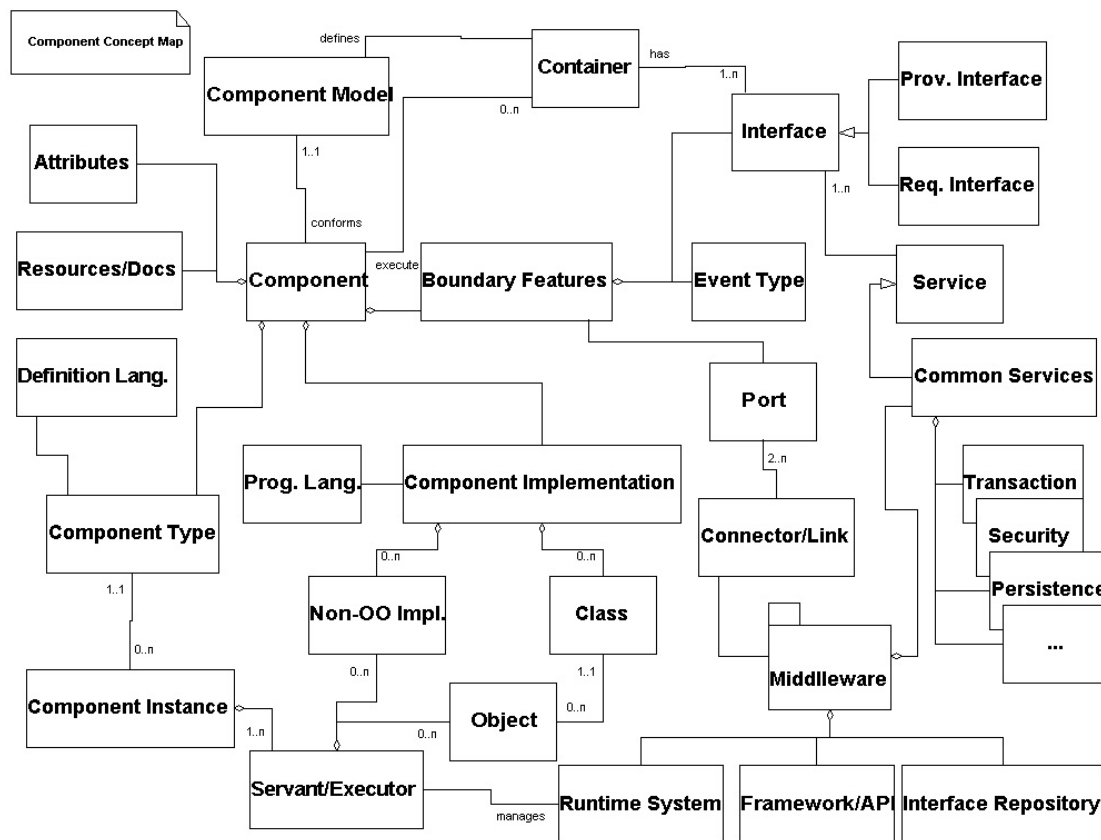


Figure 1: A Concept Map for Concepts and Their Relationships in CORBA Component Model

In CBA, interfaces play a central role in defining the roles and responsibilities of components and their interactions. The definition of interfaces is therefore the central concern of CBA. We note that interfaces can be implemented in any programming paradigms including object-oriented languages like C++ or Java.

To enable component composition at assembly time or runtime (in contrast to development time in OOA), all component interfaces support introspective operations in order to automatically discover component functionality and properties. All component models support either explicit introspective interfaces or implicit introspective mechanisms. For example, in Microsoft's Component Object Model (COM), the IUnknown interface, the root of all interfaces, is an explicit introspection interface, which includes the QueryInterface operation. Similarly, CCM uses the ComponentBase interface, which is the base interface for components to support introspection operations. JavaBeans/Enterprise JavaBeans uses implicit introspection mechanisms such as conventions and a low level reflection process as well as explicit information contained in objects implementing the BeanInfo interface.

CBA can be described in multiple views using RM-ODP viewpoints or their variations. RM-ODP defines five viewpoints: Enterprise, Information, Computation, Engineering, and Technology. The enterprise view defines the scope, purpose, policies, functional and quality requirements, as well as operation contexts. The information viewpoint defines the semantics of data objects and their relationships exchanging between components. The computation viewpoint partitions the system functionality into components and their interactions. The engineering viewpoint is concerned with the mechanisms, strategies, and patterns to support distributed interactions with respect to quality requirements. The technology viewpoint specifies the implementation technologies (platforms, languages, middleware) to achieve system functional and quality requirements. Architectural concerns in each viewpoint can be described in UML diagrams.

2.3 Service-Based Architecture

Enterprise systems face a number of challenges including seamless integration of various systems, allowing data access from anywhere anytime, and providing services to customers and partners inside and outside the enterprise. In addition to system functionality, quality considerations like extensibility, flexibility, connectivity, and interoperability also demand enterprise functions to be easily accessed via published interfaces and to be easily composed in order to offer value-added services. One way to meet these challenges and demands is to consider a system as a composition of a collection of

interacting services. Each service provides mechanism to access its functionality via well-defined interfaces. Systems developed based on such services are often structured as a kind of Service-Based Architecture (SBA).

The characteristics of SBA include

- Loose coupling. A service can receive and respond to requests from any source. The functionality of the service can be expanded or replaced, independent of its requesters. Requesters can dynamically discover and utilize services, any time and anywhere.
- Emphasizing stateless property. A service is not required to remember anything from one invocation to another. States are either stored as part of service data in database or passed from requesters. This not only helps with scalability and testability, but also allows for better maintenance.
- Services are the software units of modeling, design and implementation. A service is a software entity that encapsulates business functionality and provides the functionality to others through well-defined and published interfaces.
- Service definitions, descriptions, discovery, access protocols, and their quality aspects (e.g., security) are the central concerns in architectural design.
- Self-Describing. A service exposes its capabilities including functionality, data, and Quality of Services (QoS) characteristics through service descriptions in languages such as Web Services Description Language (WSDL).
- Discovery Mechanisms. Services can be independently and dynamically discovered and used.
- Emphasizing communication efficiency in distributed systems. Because services tend to be stateless, data exchanges between distributed requesters and providers could incur substantial overhead. To improve communication efficiency, interactions between service requesters and providers tend to be coarse-grained.

While concepts like services and service compositions have been explicitly formalized and used to deal with issues like coupling, flexibility, and plug-and-play in large-scale distributed systems in the past [10] [11] [12] [13], emerging technologies like Web Services [8] [9] and XML-based messaging made SBA more practical for large scale distributed systems. Figure 2 illustrates the key elements in a generic SBA. In SBA, a service is implemented in component-based (business components as shown in Figure 2) technologies or others like object-oriented technologies. To make it readily available for

others to use, a service is described in a service description language, commonly accepted by industry, such as WSDL. The description of the service is then registered in a service registry based on some industry commonly accepted format such as the Universal

Description, Discovery and Integration (UDDI). A customer searches for services in the registry and discovers a service it needs. Based on the service description, the customer uses the protocol and operations specified in the description to access the service.

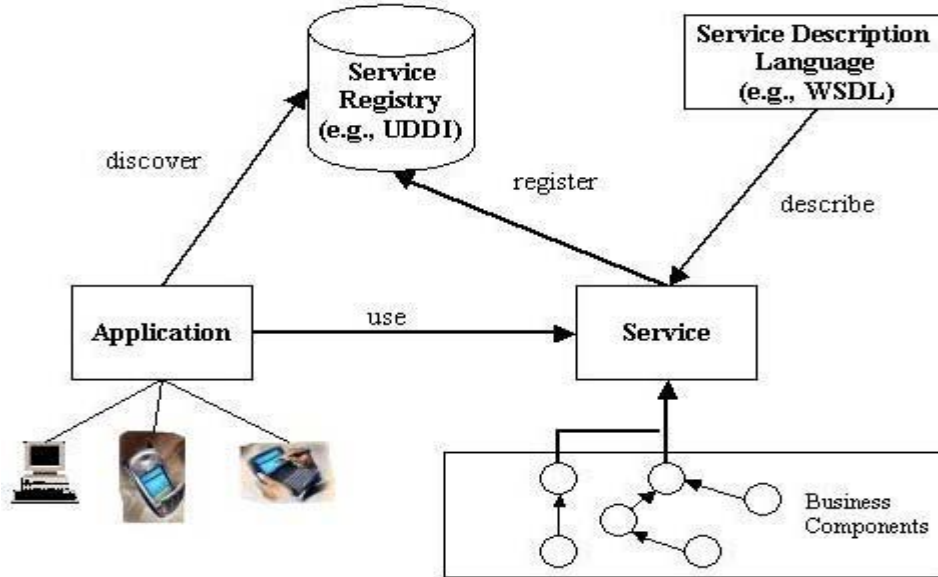


Figure 2: Key Elements in a Generic Service-Based Architecture

On the one hand, SBA can be described from the RM-ODP viewpoints (or its variations) and in UML models. On the other hand, there is no clear modeling notation in UML with regard to services. A service has many published properties which must include one or more interfaces. It may also have other properties like descriptions of protocols, entry points, and quality of service characteristics. Proposed industry standard languages like Web Services Description Language (WSDL) provide some of the notations for service providers to describe different aspects of a service.

2.4 Features and Benefits: a Comparison

OOA, CBA, and SBA have different features and benefits and may be used in a complementary manner. From an implementation perspective, a service's functionality is implemented by components and how it is implemented affects its quality properties. Similarly, a component's functionality is decomposed into one or more objects as it is implemented in an object-oriented programming language. However, these paradigms have different focuses with regard to architectural design decisions. They have different but related concepts and they have different features, benefits and concerns. Table 1 summarizes these differences.

3. Component-Based Software Systems

Along with the maturity of Component Based Software Development (CBSD) and industry component models and platforms, enterprise software systems are increasingly moving toward CBA and SBA. In the meantime, OOA becomes less appealing to large-scale distributed enterprise systems because of the concerns as summarized in Table 1. OOA remains competitive for embedded and/or real-time systems because of its support for compact and efficient implementations.

J2EE and .Net are by far the dominant industry component models and platforms for developing component-based software systems. Component-based application servers and integration servers based on such platforms become an integral part of enterprise system integration architecture. Both J2EE and .Net have been adding support for web services, therefore extending their support for developing systems using a SBA.

Figure 3 shows the roles of component-based application servers, integration servers, and web servers in web-enabled enterprise system integrations. It is a multi-tier component-based system architecture integrating backend enterprise information systems with users at the web-based front end. Business application components,

which implement business logic, are deployed in component-based application servers and business process components, which implement coordination logic in business processes, are deployed in component-based integration servers in the middle.

While an integration architecture like the one shown in Figure 3 is fully capable of integrating all kinds of enterprise systems, it does not by itself sufficient to quickly respond to changes and meet connectivity and

interoperability needs with other systems internally and externally. Quality considerations like extensibility, flexibility, connectivity, and interoperability demand a more flexible architecture paradigm. For example, value-added services must quickly and easily be built upon existing system functions and make them available to customers via intuitive self-descriptive interfaces. This flexible architecture paradigm is the SBA for component-based systems.

Table 1: Key Concepts, Features, Benefits, and Concerns of OOA, CBA, and SBA

Paradigm	OOA	CBA	SBA
Key Concepts	Class, Interface, Object Reference	Component, Interface, Container, Assembly, Introspection	Service, Service Description/Registration/Discovery, Service Composition
Focus	Identification and Definition of Classes and Their Interactions	Definition of Components and their Interfaces, Design Patterns	Definition and Description of Services and Their External/Internal Interfaces and Access Protocols
Key Features	Encapsulation of Operation and State, Object References and Method Invocations	Accepted Industry Component Models and Middleware Platforms, Introspective Interfaces, Separation of Component Development from Assembly and Deployment, Extensive Use of Design Patterns	Loose Coupling, Self Describing, Stateless
Key Benefits	Efficient, Compact, Mature Methodologies and Languages	Reduced Dependency, Reusable COTS Components, Commonality, Easy Integration, Highly Deployable, Reduced Ownership Cost	Scalable, Extensible, Flexible, Sustainable, High Connectivity and Composibility
Key Concerns	Dependencies, Tight Coupling, Small Grain as a Software Unit, Brittle to Changes with regard to both Designs and Technologies	Differences in Middleware Platforms, Support for Multiple Interaction Styles (messaging, synchronous/asynchronous, etc.), Interoperability between Heterogeneous Components, Maintenance	Quality of Services, Security, Efficiency of Interactions, Interoperability

4. Service-Based Architecture, Component-Based Systems

As an enterprise offers its business functions as services to customers, these business functions can be implemented in component-based enterprise systems as illustrated in Figure 3. Thus CBA and component based technologies can play an important role in service-based enterprise distributed systems. Similarly, proven OOA and technologies are embedded in lower layers of the system hierarchy such as middleware to take care of runtime object management and inter-object communications. It is thus clear that SBA, CBA, and OOA can co-exist in enterprise systems and they have different roles in terms of features and benefits as summarized in Table 1.

The evolution of architecture paradigms has profound impacts and influences on enterprise distributed information systems, in the ways of providing services to

customers and supporting cross-organization collaborations. In this section, we demonstrate the idea of service-based architecture, component-based systems - organizing enterprise functions as services and implementing them as component-based systems.

4.1 Component-Based Implementation of Services and Service Descriptions

A good example of services is the Weather Service from a weather forecast center, which provides historical, current, and predicated weather conditions of any particular location. Weather forecast is a complex business. It involves a lot of data processing. A query to the forecast service may require processing from a number of components of certain business logics such as data conversion and aggregation. Such a service can be implemented in a CBA as shown in Figure 3. Other

services like Map Service, Traffic Service, and Location Service can be implemented in a similar CBA.

Business components that implement services are internal to an enterprise as shown in Figure 2 and Figure 3. For example, a Map Service may ask a data access component to retrieve a map from the database then ask a location identification component to mark the location of interest on the map before presenting it to the user. Issues like where to retrieve the map, how to mark it, and which middleware platform (e.g., CORBA, J2EE, or .Net) to implement and deploy these components are some of the internal implementation issues of the map service. These issues can be dealt with in the CBA [22].

To make these services readily accessible to other applications, however, a service provider must make available explicit descriptions of their access protocols,

operations, and other properties. Furthermore, the description must be in a self-describing format like the XML-based Web Service Description Language (WSDL). By packaging business functions as services and making descriptions of these services self-explanatory, a system can be organized as a flexible and extensible SBA. New services can be added and existing services can be modified without affecting other services in the system. Clients can access these services in uniform manner. Value-added services can be created from existing services quickly. It is clear that SBA offers many advantages to enterprise systems and it has made major impacts on how enterprise functions should be implemented, packaged, and offered.

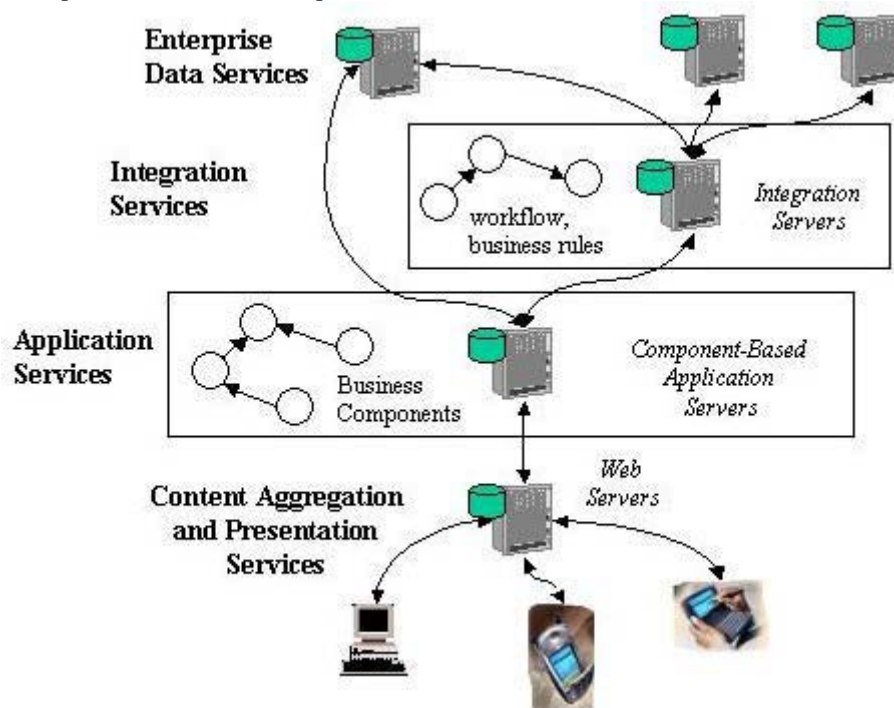


Figure 3: The Roles of Component-Based Application Servers, Integration Servers, and Web Servers in Web-Enabled Enterprise System Integrations.

4.2 Service-Based Architecture for Flexible, Extensible, and Value-Added Services

Once individual functions like finding a map and looking for driving directions, monitoring traffic conditions, knowing where you are geographically, and forecasting weather conditions are offered as Map Service, Traffic Service, Location Service, and Weather Service respectively, value-added services can be created quickly in a SBA. For example, a Real-Time Trip Planning Service can be created using these services. A

real-time trip planning service will retrieve maps, mark travel routes, select routes based on weather and traffic conditions, track customer locations in real-time, and offer directions on the spot in real-time. Similar service compositions are also possible for aviation services like a real-time flight planning service for airlines.

To offer customers secure access to a host of services, a service portal can be used to aggregate services, provides security and profile management, and offer a uniform look and feel interface to customers. Figure 4 shows how the Real-Time Trip Planning Service is created and deployed in a SBA. In this architecture, a customer logs on to the

portal from his/her favorite devices, receives its security context and necessary configuration information, and selects the service to use. Once the Trip Planning Service receives service requests, it pulls data from the Map Service, Traffic Service, Location Service, and Weather Service aggregates the trip related data, and provides real-time information to the customer. The location service

constantly receives updates on consumer movements through satellite-detected signals emitted from the consumer devices. The trip planning service correlates the location data with the map information to offer directions to the customer. Trip plans may also dynamically change depending on the weather and traffic conditions.

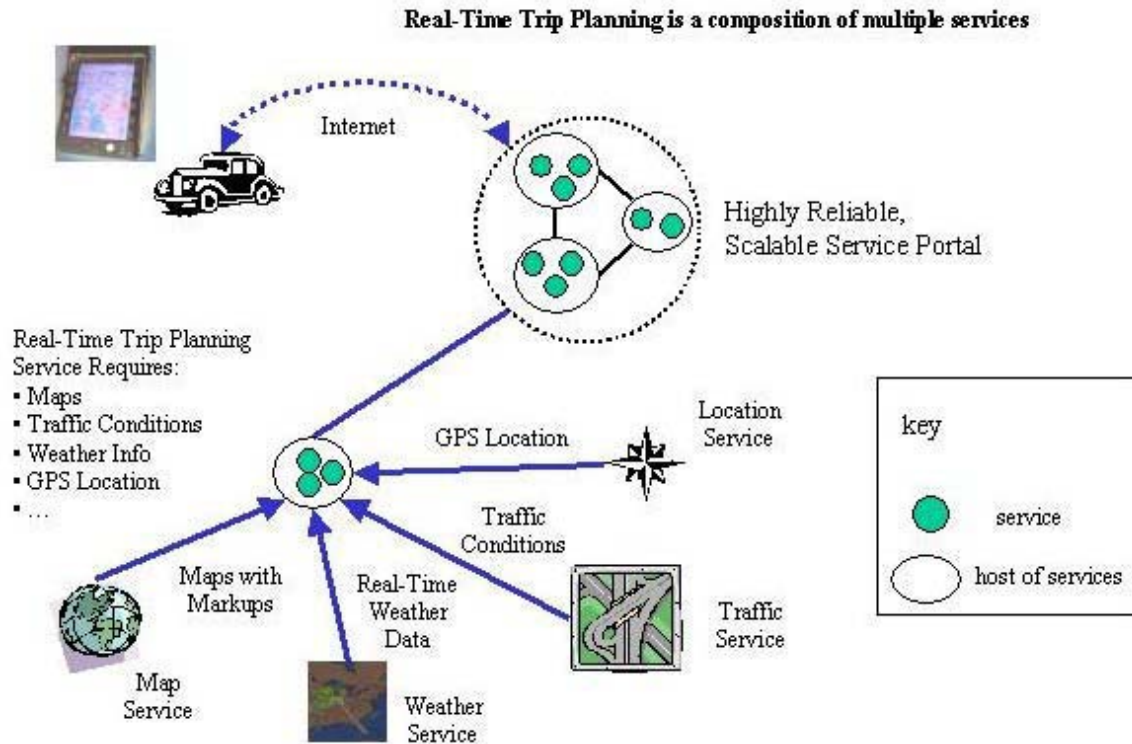


Figure 4: Real-Time Trip Planning Service is a Value-Added Service based on a Service-Based Architecture

It is possible to achieve the same real-time trip planning service using purely CBA and technologies. But it would require coordinated interface design between components in different services. It would also involve static binding between components so that a component can invoke operations of another based on the interfaces. The disadvantages of this solution include being not flexible to accommodate changes, hard to extend to new business functions, and difficult to maintain. Similarly, it is possible to achieve the same real-time trip planning service using purely OOA and technologies. This solution has similar disadvantages as the CBA approach. In addition, it requires complex and often proprietary operations to invoke methods of remote objects in distributed systems.

5. Practical Principles and Strategies

Object, component, and service are three key software development concepts. Software development methodologies have been evolving over these concepts. With advances in technologies like the World Wide Web (WWW) and the constant search for more effective enterprise information services to internal and external customers and partners, we are witnessing demand for more open, flexible, and extensible architectures and solutions for enterprise systems. Software architecture paradigms have also evolved from OOA, CBA, to SBA. Industrial specifications of object-oriented modeling languages, component-based middleware platforms, web services, and XML-based data representations provide the technical foundation for the implementations of distributed systems based on OOA, CBA and SBA.

In the evolution of the architecture paradigms, the role of an architecture paradigm and related technologies in enterprise distributed applications has changed. OOA is fundamental to detailed system designs and implementations based on object-oriented modeling tools and programming languages. In particular, OOA and technologies are the foundation of low level distributed system infrastructures to support object-oriented interactions. CBA, on the other hand, is fundamental to the development and deployment of components of business logic in the mid-tiers of enterprise architectures (see Figure 3), application servers and integration servers in particular. Customer oriented (internal or external) business functions are increasingly being organized into software services. A service is implemented directly by components in the mid-tiers or a value-added service composed from other services (see Figure 4). SBA and related technologies play an essential role for building open, flexible and extensible services.

Each of the architecture paradigms has its unique concepts, characteristics, features, and benefits. Separation of concerns, modularity, abstraction, anticipation of change, and evaluation of design alternatives are some of the general software engineering principles useful in designing an application architecture based on one of the three paradigms. Because every one of the paradigms has embodied these principles, we need additional practical principles to guide us to make sound architectural design decisions. We can derive three principles from the software engineering practices:

- The first principle is to understand the Purpose and Scope of the application. The purpose and scope define the intended customers, boundaries with other applications, operating environments, and properties of application domains. This principle requires us to understand issues like what the customers want, what constraints the operating environments have, and how the application intends to interact with others.
- The second principle is to know the Desired Quality Attributes, particularly the performance, flexibility, extensibility, sustainability, openness, and interoperability, from the stakeholders. The desired quality attributes are then matched against the features and benefits of the architecture paradigms as summarized in Table 1.
- The third principle is to define Use Case Scenarios for Interfaces. Interface is a key concept common to all of the paradigms, yet it is independent of the core concept of Object, Component, and Service, respectively in each of them. A software unit, an object, a component, or a service implements interfaces. Use case scenarios for interfaces help clarify the roles,

responsibilities, interaction characteristics, and expected properties of the software units and thus help identify which of the three types of software units is best suited for the application architecture.

Given these principles, a practical strategy for sound architecture decisions is to decide an architecture paradigm based on the purpose and scope of the application, the desired quality attributes from stakeholders, and the use case scenarios of interfaces. At the same time, it should be recognized that OOA, CBA, and SBA are not mutually exclusive and they are complementary in many aspects. For example, an enterprise's business functions oriented toward customers/partners (internal or external) can be organized in an SBA, while these services can be implemented as component-based systems based on the CBA.

6. Conclusions and Future Work

In this paper, we discussed the influences and impacts of architecture paradigms, namely OOA, CBA, and SBA, on distributed component-based software systems. We described the concepts and concerns in each of the paradigms and compared and contrasted their characteristics. Component-based system developments are discussed in the context of enterprise systems integration needs. We demonstrated the benefits of SBA in its openness, flexibility, and extensibility using the Real-Time Trip Planning Service as an example. Alternate implementations of the service based on OOA and CBA are also discussed. Some practical principles and strategies are derived for best utilizing appropriate architecture paradigms in software engineering practices.

It is important to recognize that different architecture paradigms have different characteristics, features, and benefits. The role of OOA and related technologies has shifted to detailed implementations in lower layers of distributed enterprise systems. CBA and related technologies are most effective for developing and deploying business components in mid-tiers of enterprise systems, application servers and integration servers in particular. For customer/partner (internal and external) serving business functions, they are best organized as services in a SBA to take advantage of its openness, flexibility, and extensibility as well as its simplicity for developing value-added services.

Our future work is around the development of architectural frameworks for deriving and specifying architectural products in software engineering processes for large-scale systems integration. Large-scale system architectures tend to be complex and require descriptions from multiple views in multiple levels of abstractions [14] [15]. Each of the descriptions is an architectural product.

An architectural framework is critical to define precisely what products are necessary, how to describe them, and how to tie them together to provide a complete description of the system architecture. Such an architectural framework must take into consideration of architecture paradigms, component-based technologies, and object-oriented modeling notations (in UML) as discussed in the paper.

7. References

- [1] Ivar Jacobson, Grady Booch, James Rumbaugh, *The Unified Software Development Process*, Addison-Wesley, 1998.
- [2] Phillippe Kruchten, "The 4+1 View Model of Architecture", *IEEE Software*, Vol. 12, No. 6, November 1995.
- [3] Sun Microsystems, Java 2 Platform, Enterprise Edition (J2EE), <http://java.sun.com/j2ee/>
- [4] Object Management Group (OMG), CORBA Components Specification, OMG document number: orbos/99-02-05 (<http://www.omg.org>).
- [5] Object Management Group (OMG), Common Object Request Broker Architecture (CORBA) specification, version 3.0, 2002, (<http://www.omg.org>).
- [6] Object Management Group (OMG), UML 2.0 specification, 2003, (<http://www.omg.org>).
- [7] Keith Ballinger, *.NET Web Services: Architecture and Implementation*, Pearson Education, November 2002.
- [8] Ali Arsanjani, Brent Hailpern, Joanne Martin, Peri Tarr, "Web Services: Promises and Compromises", *ACM Queue*, March 2003, pp. 48-58.
- [9] E. Newcomer, *Understanding Web Services: XML, WSDL, SOAP, and UDDI*, Pearson Education, 2002.
- [10] G. Wang, L. Ungar, D. Klawitter, "Component Assembly for Object-Oriented Distributed Systems", *IEEE Computer*, July 1999, pp. 71-78.
- [11] G. Wang, H. A. MacLean, "Software Components in Contexts and Service Negotiations", *CBSE workshop 1999*, May 1999.
- [12] G. Wang, L. Ungar, D. Klawitter, "A Framework Supporting Component Assembly for Distributed Systems", *Proceedings of the Second Enterprise Distributed Object Computing Conference*, La Jolla, CA. IEEE CS Press, Nov. 1998, pp. 136-146.
- [13] G. Wang, H. A. MacLean, "Architectural Components and Object-Oriented Implementations", *CBSE workshop 1998*, April 1998.
- [14] G. Wang, G. Cone, "A Method to Reduce Risks in Building Distributed Enterprise Systems", *Proceedings of the Fifth IEEE International Enterprise Distributed Object Computing Conference*, IEEE CS Press, September 2001, pp. 164-168.
- [15] G. Wang, G. Cone, "Architecture Reuse from Multiple Levels of Abstraction and Multiple Viewpoints", *4th World Multiconference on Systemics, Cybernetics and Informatics, (SCI'2000)*, July 23-26, 2000, Orlando, FL., pp. 619-624.
- [16] Open Distributed Processing – Reference Model, URL: <http://www.iso.ch:8000/RM-ODP>.
- [17] IEEE Std 1471-2000, IEEE Recommended Practice for Architectural Description of Software Intensive Systems, 2000, <http://standards.ieee.org/catalog/software4.html#1471-200>
- [18] Colin Atkinson, Joachim Bayer, Christian Bunse, Erik Kamsties, Oliver Laitenberger, Roland Laqua, Dirk Muthig, Barbara Peach, Jurgen Wust, Jorg Zettel, *Component-Based Product Line Engineering with UML*, Addison Wesley, 2001.
- [19] Desmond D'Souza, Alan Wills, *Objects, Components, and Frameworks with UML, The Catalysis Approach*, Addison Wesley, 1998.
- [20] Clemens Szyperski, *Component Software: Beyond Object-Oriented Programming*, Addison Wesley, 2002.
- [21] Alan Brown, Kurt Wallnau, "The Current State of CBSE", *IEEE Software*, Oct. 1998, pp. 37–46.
- [22] A. Brown, S. Johnston, K. Kelly, "Using Service-Oriented Architecture and Component-Based Development to Build Web Service Applications", *Rational Software White Paper*, October 2002