MODELING AND ANALYZING SERVICE-ORIENTED ENTERPRISE ARCHITECTURAL STYLES

by

Longji Tang

APPROVED BY SUPERVISORY COMMITTEE:

To my wife Jianglan Hu and my daughter Helen Tang Paradise

MODELING AND ANALYZING SERVICE-ORIENTED ENTERPRISE ARCHITECTURAL STYLES

by

LONGJI TANG, B.E., M.S.

DISSERTATION

Presented to the Faculty of

The University of Texas at Dallas

in Partial Fulfillment

of the Requirements

for the Degree of

DOCTOR OF PHILOSOPHY IN COMPUTER SCIENCE

MAJOR IN SOFTWARE ENGINEERING

THE UNIVERSITY OF TEXAS AT DALLAS

December 2011

PREFACE

This dissertation was produced in accordance with guidelines which permit the inclusion as part of the dissertation the text of an original paper or papers submitted for publication. The dissertation must still conform to all other requirements explained in the "Guide for the Preparation of Master's Theses and Doctoral Dissertations at The University of Texas at Dallas." It must include a comprehensive abstract, a full introduction and literature review and a final overall conclusion. Additional material (procedural and design data as well as descriptions of equipment) must be provided in sufficient detail to allow a clear and precise judgment to be made of the importance and originality of the research reported.

It is acceptable for this dissertation to include as chapters authentic copies of papers already published, provided these meet type size, margin and legibility requirements. In such cases, connecting texts which provide logical bridges between different manuscripts are mandatory. Where the student is not the sole author of a manuscript, the student is required to make an explicit statement in the introductory material to that manuscript describing the student's contribution to the work and acknowledging the contribution of the other author(s). The signatures of the Supervising Committee which precede all other material in the dissertation attest to the accuracy of this statement.

ACKNOWLEDGEMENTS

It has been a great pleasure working with the faculty, staff, and students in The University of Texas at Dallas, during my studies as a doctoral student on a part-time schedule. This work would never have been possible if it were not for the freedom I was given to pursue my own research interests. I would like to give appreciation to many people for their inspiration and encouragement that led to the completion of this dissertation.

I owe my deepest gratitude to my Ph.D. program advisors, Dr. Farokh B. Bastani and Dr. Jing Dong. I never could have completed this dissertation without their dedicated instruction and assistance.

Specifically, I greatly appreciate Prof. Wei-Tek Tsai of School of Computing, Information and Decision Systems Engineering at Arizona State University for his kindness and considerable guidance on my research and dissertation.

It is an honor for me to have Dr. Kang Zhang, Dr. Lawrence Chung, Dr. Jing Dong and Dr. Wei-Tek Tsai as my dissertation committee members. I am grateful for their provision of a great number of valuable comments on my work.

I would like to thank my company FedEx for providing me full financial support through the FedEx education program. I greatly appreciate my vice presidents, manager directors, managers, and my colleagues for their encouragement and support.

I owe my thanks to Dr. Yajing Zhao and Peng Tu for their collaborations on my research work.

This dissertation would not have been possible without my family and other friends' continuous support. I particularly thank my wife, Jianglan Hu, and my daughter, Helen Tang Paradise, for their continuous encouragement.

September 2011

MODELING AND ANALYZING SERVICE-ORIENTED ENTERPRISE

ARCHITECTURAL STYLES

Publication No. ___________________

Longji Tang, Ph.D.
The University of Texas at Dallas, 2011

Supervising Professor: Dr. Farokh B. Bastani

Modern enterprises consist of complex business systems. The Enterprise Service-Oriented Architecture (ESOA) becomes an important enterprise architectural style (EAS) for designing and implementing business systems. Cloud computing is a new paradigm of distributed computing and is bringing many new ideas, concepts, principles, technologies, and architectural styles into enterprise service-oriented computing. A new hybrid architectural style with ESOA and cloud computing, Enterprise Cloud Service Architecture (ECSA), is emerging as the future design principle of service-oriented enterprise architecture. The methodology and design principles of Service Level Agreement (SLA) and Quality of Service (QoS), originally used in telecommunication and networking services, are increasingly being adopted in enterprise service computing. Combining SLA and QoS with ESOA and ECSA is forging new kinds of service-oriented enterprise architectural styles: SLA-Aware ESOA and SLA-Aware ECSA. To better understand enterprise service-oriented architectural styles and guide their system specification, design, implementation, and runtime behavior, a framework for specifying and analyzing service-oriented enterprise architectural styles is needed. The software architectural style is an abstraction of a family of concrete software architectures. It specifies the architectural structures and includes elements and connectors, design principles, and system constraints as well as non-functional behavior.

The research work in this Dissertation is motivated by the desire to understand and evaluate the architectural design of enterprise service-oriented architectural applications in terms of ESOA and ECSA structures, principles, and constraints, thereby achieving higher architectural quality, including enhanced performance, scalability, security, and other quality attributes of the architecture. This Dissertation defines a framework based on EAS ontology for modeling and analyzing service-oriented enterprise architectural styles and its various extensions, refinements, and compositions both formally and informally. The framework not only specifies the generic structures of ESOA and ECSA systems, but also specifies system constraints through software architectural quality attributes. The framework emphasizes performance, security, elastic scalability, dynamic infrastructure, tradeoff of quality attributes, and enterprise service-oriented system runtime management. With the framework, the Dissertation presents models and analyzes ESOA and ECSA styles as well as their extensions and compositions, such as SLA-Aware ESOA and SLA-Aware ECSA. The consistency of properties, constraints, and refinements of the service-oriented enterprise architectural styles are formally and informally analyzed. This is used to understand and evaluate service-oriented enterprise architectures and provide guidance for the design of ESOA and ECSA systems as well as their SLA-Aware systems. Moreover, the dissertation describes and discusses the lessons learned from various ESOA and ECSA style architectures.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

CHAPTER 1

INTRODUCTION

The general software architectural style has been used to specify various architectural families and guide software system design since Perry and Wolf [170] and Shaw and Garlan [185] introduced the concept of software architecture. The complexity of modern enterprise information systems is evolving the architecture of software systems from component-oriented styles to service-oriented styles, where services are the central and basic elements in specific architectural systems. The services are designed and implemented for communicating with their consumers through a network and performing various business tasks. With the Internet becoming a global information superhighway and a common platform that spans across enterprises and their geographic locations, the enterprise information architecture is becoming ubiquitous and continuing its evolution from on-premise service-oriented styles to cloud service computing styles. The evolution of enterprise architectures brings forth opportunities for modern software architecture research. Software architecture research studies methods of determining how best to construct an enterprise system, how services identify and communicate with each other, how information and data are communicated, how an enterprise system can meet software quality constraints, and how all of the above can be specified by using formal and informal notations. Software architectural style specifies the common architectural elements, design patterns, principles, and common constraints for a family of specific architectures in terms of certain formal and informal notations. Hence, research involving the investigation of architectural styles can help both architects and practitioners in understanding and evaluating complex systems and guiding the correct design decisions.

As a new software architecture style, service-oriented architecture (SOA) [62][66] [213][215][204][178] is receiving significant attention. SOA promotes loosely coupled architecture, interoperability, reusability, and extensibility. Due to the globalization of economic environment, business processes are becoming more and more complex, which makes enterprise information systems more and more complex. Enterprise service-oriented architecture (ESOA), a new architectural style, is designed to help enterprises to build better architectures and solutions for serving the increasingly sophisticated business processes. Conceptually, the ESOA is an architectural style which defines any concrete ESOA architecture as a set of well-defined services. It may be further abstracted to process layers and composite applications for business solutions. The services are deployed and accessed through the SOA infrastructure. They are governed and managed by SOA principles and management system. Enterprise Service-Oriented Architecture (ESOA) is a specific style of SOA, which has been used in many industrial applications [112]. The ESOA and its substyles [201][202][203][204] define the common service-oriented architectural elements, design principles, and a coordinated set of constraints – functional and non-functional for various enterprise architectures. Therefore, research in ESOA and its substyles can uncover generic models of service-oriented enterprise systems and determine a system’s overall properties which can be defined by architectural quality attributes and the service properties. This results in a better high-level architectural understanding and enables correct design decisions as well as the selection of appropriate technologies for a given system.

The ESOA brings an agility aspect to enterprise architecture, allowing enterprises to deal with system changes using a configuration mediation layer, rather than constantly having to redevelop these systems. However, ESOA introduces new challenges and issues to enterprise architecture because of its following on-premise characteristics:

- The enterprise owns its data center with ESOA services and the infrastructure is not dynamic, such as not supporting auto scaling and elastic load balancing [10].
- The enterprise architecture is built behind firewalls.
- The resources are dedicated to each workload.
- The resources are shared within the enterprise only.

Building a data center to support ESOA architecture is expensive and is not possible for some small to medium enterprises. For large enterprises, it is not possible to complete some complex business processes, such as online shopping and shipping, without third party services. Moreover many server resources in a large data center are idle or passive, such as during non-peak times, since the acquisition of resources is based on the need to be able to cope with peak workloads. Thus, resources are wasted, thereby resulting in increasing cost of resources and operations. Many enterprises view SOA as something that only occurs within firewall. The ESOA is facing new challenges from enterprises – reducing complexity as well as cost and increasing capacity, flexibility as well as agility. Cloud computing as a new paradigm of distributed computing is being applied to enterprises, which brings forth many new ideas, concepts, solutions, principles to enterprise architecture and ESOA. Originally, cloud computing evolved from web computing (such as web 2.0 [83]), service-oriented computing [244][215][201][204], grid computing [246], utility computing [35] and other technologies –

virtualization [98] and virtual applications. Cloud computing is about moving services, computation, and/or data to an on-premise or off-premise, location-transparent, centralized facility or contractor for cost and business advantages. By making services and data available in the cloud, it can be more easily and ubiquitously accessed, often at much lower cost, thereby increasing its value by enabling opportunities for enhanced collaboration, integration, and analysis on a shared common platform [52]. On the other hand, cloud computing without adopting ESOA's service orientation, service management, and other SOA principles, will most probably fail and not be adopted by enterprises. Therefore, combining cloud computing and ESOA takes ESOA to the next level and expands it from on-premise to off-premise.

This dissertation investigates service-oriented enterprise architectural styles (SOEAS) which include ESOA and a junction on the frontiers of two new software architectural styles in enterprise distributed computing, namely, enterprise service-oriented architectural style and enterprise cloud computing style. The intersection of these two paradigms leads to a new architectural style, Enterprise Cloud Service Architecture (ECSA). The ESOA has been investigated and practiced for several years in both academia and industry. However, generic understanding and categorization of models and substyles are still lacking. There are many challenges from enterprises and software industries. In contrast, as an emerging technology, enterprise cloud computing becomes a key to improving ESOA systems in both academia and industry. My research work is motivated by the aspiration to (i) better understand both new architectural styles and their intersection ECSA, (ii) evaluate the architectural design by specifying a generic model of the ESOA and ECSA, and (iii) guide the process of decision making for designing ESOA and ECSA systems with high quality assurance in both structural and behavioral aspects.

To achieve high quality assurance in ESOA and ECSA style systems, the technology of Service Level Agreement (SLA) and Quality of Service (QoS), originally from telecommunication and networking services, are adopted by service-oriented enterprise architecture. Also, the Service Level Management (SLM) is becoming an important design methodology and principle in ESOA and ECSA. The dynamic SLM provides an SLA-Aware approach in an ESOA or ECSA architecture. Therefore, adding SLA-Aware into ESOA and ECSA generates two new enterprise architecture styles, namely, SLA-Aware ESOA and SLA-Aware ECSA. Finally, the dissertation investigates the new important enterprise architectural style based on the proposed ESOA and ECSA models.

The rest of the dissertation is organized as follows: Chapter 2 presents a review of the existing work related to this research. In particular, research regarding general architectural styles as well as the ESOA and enterprise cloud computing literatures are surveyed and classified. Chapter 3 builds a framework for modeling and analyzing service-oriented enterprise architectural styles. The framework is based on architectural style ontology [168] and the model proposed in [201][204][205]. Chapter 4 includes (1) a proposed model of ESOA and specifications of two major substyles of ESOA by the model; (2) classification of five major substyles of ESOA. Chapter 5 models and analyzes a new hybrid enterprise architectural style ECSA. Chapter 6 introduces the SLA-Aware enterprise service computing and specifies new styles SLA-Aware ESOA and SLA-Aware ECSA. Chapter 7 Analyzes the ESOA and ECSA based on the framework proposed in Chapter 3 and defines ESOA and ECSA system evaluation method and discusses several case studies.

In summary, this dissertation makes the following contributions to software architecture research within the field of Software Engineering:

- It defines a framework for modeling and analyzing ESOA and ECSA styles;
- It presents a classification of ESOA substyles;
- It extends ESOA and ECSA models for specifying the new styles: SLA-Aware ESOA and ECSA;
- It develops methods for evaluating ESOA and ECSA systems;
- It presents applications and a detailed evaluation of ESOA and ECSA systems.

CHAPTER 2

RELATED WORK

There have been numerous studies on research of general software architectural styles and enterprise service-oriented architecture. Although enterprise cloud computing (ECC) is an emerging area, the research work of ECC is increasing significantly. The SLA-Aware SOEA is getting more and more attention because of the enterprise QoS challenges. This chapter provides a comprehensive review of the literatures.

2.1 Software Architecture and Architectural Styles

Currently, software architecture is being increasingly recognized as an important part of the software engineering discipline. Over the past two decades, software architecture research has emerged as the principal basis for specifying the overall high-level structure and relationships among subsystems and components of software systems [184][183], especially the software quality attributes of systems, which can be best designed and analyzed at the system level [114]. Software architecture research focus on the following two aspects:

(1) Specification of software architecture, which includes formal and informal architectural modeling and representation;

(2) Analysis and evaluation of software architecture, which include analyzing and evaluating its structure and its quality attributes (or properties) formally and informally.

Modern software architecture research, especially enterprise software architecture research, has greatly increased its interest in software architecture integration, dynamism and automation of systems [209].

There are many definitions of software architecture [24][49][185][209]. This dissertation focuses on enterprise software architecture research. We define this as follows:

An enterprise software architecture is an abstraction of enterprise-level software systems. It consists of the set of principal design decisions (*PDD)* and the corresponding set of architectural artifacts (*AA)* made about the enterprise IT systems.

Mature common software architectural principles, structure, and description are often applied to a family of systems repeatedly. This leads to software architectural styles, such as pipe-filter, client-server [185]. Software architectural style is increasingly getting attention from both software engineering researchers and practitioners. With the growing complexity of modern software systems, specifically large distributed enterprise systems, architectural style is becoming increasingly important for achieving high quality software architecture design and enabling accurate system quality analysis. Hence, research regarding software architectural styles is greatly inspired by its value to the design of cost-effective enterprise software architectures. Based on some traditional definitions of architectural styles [185][181][209][49], we propose an informal definition of enterprise architectural style as follows:

An enterprise architecture style is an abstraction of a family of enterprise-level software systems. It consists of the set of architecture design principles and architectural quality attributes, and the set of descriptions of common structures and behaviors, and common constraints. It guides how they can be applied to form a concrete enterprise architecture that meets enterprise architecture requirements.

Compared with software architecture, architectural style is the parent of a family of concrete architectures and concrete architecture is an instance of its architectural style. Similar to software architecture research, research of software architectural styles mainly focus on the following two aspects:

(1) Specification of architectural style, which includes formal and informal modeling and representation of architectural styles.

(2) Analysis and evaluation of architectural style, which includes its structure as well as behavior, consistency as well as correctness, and composition as well as instantiation.

In this section, the related research work on these two aspects are summarized and reviewed.

2.1.1 Traditional Architectural Style Modeling

Software architecture as a software engineering discipline has been investigated by researchers since the early 1990s. The development of software architectural models and styles was pioneered by Perry and Wolf in their article on “Foundations for the Study of Software Architecture” [170]. This article introduced a formula:

$$\text{Software Architecture} = \{\text{Elements, Form, Rational}\}, \qquad (2.1)$$

and it defined the architectural style as an abstraction of specific architectures. Boehm added "constraints" to (2.1) shortly thereafter [114] and Shaw and Garlan [180][185] describe the software architectural style as a family of systems in terms of a pattern of structural organization. Several typical architectural styles are specified in [185] formally and informally. Abowd, Allen and Garlan give a formal definition of architectural style and use formalized architectural styles to understand the description of software architectures [2]. The early research works [114][180][185] of software architectural styles focused on building basic concepts as well as developing foundations and specifying common traditional styles, such as pipes and filters and client-server patterns. The early research work was also evolved by the software engineering community into what are called architectural patterns [241].

Shaw and Garlan define architectural style as "a vocabulary of components and connector types, and a set of constraints on how they can be combined" [185]. Traditional architectural models and concepts of architectural style are fundamental to our research on ESOA models and styles. However, traditional architectural style [114][180][185] is based on three main parts in software architecture :

- Components (Elements - Perry/Wolf style);
- Connector types;
- Constraints, such as a set of configuration rules.

Klein and Kazman developed the quality attribute-based architectural styles (ABAS) [158][159] which are used to aid in the design of architecture for large, complex systems and are also used in analyzing existing systems as part of the Architecture Tradeoff Analysis Method (ATAM) [158][159]. My partial research work is also motivated by the ABAS and ATAM.

Bass, Clements and Kazman have further developed architectural styles for modern component-oriented software systems, such as COM, CORBA and J2EE/EJB systems in [24].

Fielding and Taylor proposed and evolved the Web (World Wide Web) architectural style called as Representational State Transfer (REST) in [70]. Khare and Taylor extended the REST style to several new styles for decentralized systems [107]. The REST is a basis for one of the ESOA sub-styles we specify later.

Singh, et al., propose an SOA model – Commitment-Based Service-Oriented Architecture (CSOA) in [187]. CSOA defines components as business services and connectors as patterns, modeled as commitments which support key elements of service engagements. Although our approach for modeling ESOA styles is very different from [187], the ESOA specifies services as enterprise services which are based on customers' functional and non-functional requirements. Moreover, we consider quality attributes as constraints of ESOA styles, which are similar to the CSOA commitments.

In the past decade, component-based architecture systems have evolved to service-oriented architecture systems, especially in many enterprises with necessities of resolving various business demands, such as better integration, better agility, and better quality. The architectural styles are developed from component-based and object-oriented styles to service-oriented architectural styles (SOA) [127] and ESOA [201][202][204] which is a specific SOA style for enterprise. Nowadays, the enterprise cloud computing [125][243] as a new distributed style is receiving a great amount of attention by researchers in both industry and academia. My research work focuses on ESOA styles as well as ECC styles and their intersection. In the next two sub-sections, these are surveyed and classified.

2.1.2 Formalism and Informalism for Architectural Styles

Except for some traditional and simple architectural styles, such as pipe-filter and client-server styles, the current specification and modeling of architectural styles is informal and ad hoc [6]. As we know, formalism of software architecture has been investigated for two decades. Many formal methods [181], like various Architectural Description Languages (ADL), such as ACME [79], Rapide [78] and Wright [78][200], have been developed. The formalism of architectural styles, especially of enterprise architectural styles, is still in its early stages. Because of the great complexity of modern software architectures, such as WWW architectures and large enterprise architectures, pure formalism of architectural styles is not always possible [209]. R. Allen showed in [6] that both formalism and informalism of architectural style have their advantages and disadvantages as listed in Table 2.1:

Table 2.1. Comparison of Formalism and Informalism

Formalism	Informalism
• Precise	• Easy to understand
• Provable properties	• Shows how to build one
• Structures analysis	• Structures design
• Based on architectural principles	• Based on architectural intuitions

The conclusion is that formal and informal methods are both useful and needed for specifying architectural styles [6] [209].

Formal modeling of architectural styles can be classified into the following major methods as shown in Table 2.3:

Table 2.2. Formal Modeling Methods of Architectural Styles

Formal Modeling Method	Characteristics of method	References
ADL-based methods, such as ACME, Alloy	• Using extendable ADL for modeling architectural style • Describe styles in terms of basic ADL syntax as well as semantic and type concept o A set of structure types – component types, connector types and system topology (configurations) which provide the architecture structure design vocabulary o A set of property types which provide the semantic vocabulary for a family of system. o A set of constraints which decide how style's instances of those types can be used. o A default structure which prescribes the minimal set of instances that must appear in any system of the style. • Tool support – AcmeStudio and Alloy Applyzer • Based-on first-order logic • Can precisely describe traditional component-based architectural styles, such as pipe-filter and client-server • Does not support architecture dynamism and makes it hard to specify large distributed systems' styles	ACME [77][79] Alloy [108]
Process Algebra-based methods, such as PADL	• Formalizing architectural styles by means of a process algebra based ADL – PADL whose syntax and semantics are based on process algebra • Describing styles by introducing the intermediate abstraction of architectural type in a process algebra framework. • Provide type checking for evaluating architectural compatibility • Provide precious formalism for traditional architectural styles, such as pipe-filter.	PADL [25]
LOTOS-based methods	• LOTOS consists of two parts – an algebra specification language for defining data and a process algebra for defining the system behavior. • Combining LOTOS pattern with constraints can specify architectural style formally. • Some of traditional styles, such as shared memory, pipe-filter, are specified by LOTOS	LOTOS [88]
Z and CHAM-based methods	• Z formal notation is a descriptive language which can be used for modeling structure of architectural styles • CHAM = Chemical Abstract Machine allows us to model and analyze dynamic properties of architectural style • Mapping Z notation to a formal operational semantics based on CHAM can allow us to specify and analyze architectural style using tools.	Z + CHAM [47]
Graph-based methods	• Its formalism is based on formal graph theory. • In [93], software architectural style is defined in terms of graph (Nodes, Edges, Grammar), in which Nodes represent components; Edges denote the interconnection of components. The graph grammar defines style. • In [139], software architectural style is described as a hyperedge context-free grammar. • Both describe traditional client-server style • Both provide dynamic reconfiguration, refinement through graph rewriting	[139][93]
Ontology-based methods	• Ontology-based approach represents architectural style as architectural knowledge (vocabulary) which is formalized based on description logic. • It is easy to integrate with some ADLs and other modeling languages, such as ACME for modeling and analyzing architectural styles. • Traditional architectural styles can be described and analyzed by the approach.	[167][168]

From Table 2.2, we can see that the main advantage of the formal approach is its high precision in accurately modeling and analyzing architectural styles since it provides description languages for specifying each style's structure and behavior. However, formal methods are hard to understand and lack the ability to specify large and complex enterprise architectural styles, especially in modeling and analyzing their dynamic behaviors and changes. In the next section, we will present a few ADLs that can describe dynamism in software architectures to at least a limited extent and a few formal methods in Table 2.2 have features for specifying dynamic behavior and changes. Moreover, the dissertation research shows that combining formal and informal approaches (FINF method) is a way to model and analyze modern complex distributed software architectural styles, especially enterprise IT architectural styles in today's dynamic environment, such as enterprise cloud service architecture (ECSA) [205].

In fact, the approach in [170] is the first FINF method for specifying software architectures and architectural styles. SEI Attribute-Based Modeling of Architectural Styles [110] is another typical FINF approach for specifying and analyzing architectural styles. The research on software architectural styles developed by Prof. Taylor and his students [209][71][70][107] adopted the FINF approach. Unlike earlier UML versions, UML 2.0 has been developed with some capabilities for describing software architecture as a semi-formal ADL. Its extension capacity, namely, UML profile plus OCL (Object Constraint Language), allows the description of some of the architectural styles [49][209]. The biggest advantages of the UML approach are (1) it can virtually describe software architecture, (2) it has tool support, and (3) its notations are easy to understand by software engineers and architects. It is good at describing static aspects, such as the structure (components and connectors) of the architecture and its style and some limited dynamic behaviors by its behavior diagrams, such as activity diagram, state diagram, and sequence diagram. However, it lacks the capability to describe dynamic infrastructures and architectural quality attributes as well as their tradeoffs.

2.1.3 Modeling Dynamism of Software Architecture and its Styles

Dynamism is playing more and more important roles in modern software architectures and architectural styles, such as enterprise SOA and cloud service computing, and is receiving increasing attention of software architecture researchers. Dynamism is a kind of characteristics and measure of dynamic software system, such as dynamic scalability. Specifying and modeling dynamism of software architecture and architectural style is not easy. Only a few ADLs have some capability to describe dynamism. Darwin [209] only allows constrained dynamism. Rapide [209] can express event-based dynamic architecture. Process Algebra-based formal languages, such as CHAM [47], CSP and Pi-Calculus [200][209], are good at describing dynamic behavior, especially such as communication and interaction activities.

Not all architectural styles are capable of supporting the specification of dynamic reconfiguration and dynamic system quality attributes. C2 [209] is a traditional architectural style which facilitates runtime reconfiguration. Some formal modeling methods, such as PADL [25] and graph-based methods [139][93], are able to describe dynamism of architectural styles. Enterprise SOA and cloud computing are highly dynamic architectural styles. The above-mentioned modeling methods have some limitations, especially because they lack capabilities for modeling dynamic service-orientation and supporting dynamic quality attributes. Specifying and analyzing the dynamism of ESOA and ECSA is an important part of this dissertation.

2.1.4 Architectural Pattern, Type and Styles

In the early 1990s, software architecture research introduced architectural style concept [180][181][185] which is based on the observation of recurring coarse-grained problem solutions in related systems (or family of systems), which use a set of specific elements with certain relationships. In parallel with the research on architectural styles, design patterns and pattern languages have been used for describing common design solutions or idioms that are found repeatedly in object-oriented software systems. R.T. Monroe, et al., compared architectural styles with design patterns and pointed out that they are related in two ways in [145]:

- Architectural styles can be viewed as kinds of architecture design patterns or pattern languages.
- A given architectural style may use a set of idioms which can be viewed as micro-architectures or design patterns.

P. Clements, et al., describe commonalities and differences in both architectural styles and patterns [49]. The common goal of both architectural style and pattern research is not to make up solutions but to capture solutions that are already in use. However, there is a slight difference between architectural styles and patterns. In particular, a style tends to refer to a coarse grain design solution (or decision) for a family of systems while a pattern tends to refer to a design solution localized within a few (or one of many) architectural components of a system [49]. L. Bass, et al., propose that an architectural pattern is equal to an architectural style [24]. In this dissertation, we focus on research of architectural styles.

As is well known, the traditional ADL ACME [77][79] can be extended to specify architectural styles via its type system. In ACME, three types can be defined by architects – ***property types***, ***structure types*** and ***family types (styles)***, which can be used to encapsulate recurring structures and relationships in a family of systems. The structure types can help define component types, connector types and ports, and roles. Each component type and connector type defines its name and a list of required substructures, properties, and constraints. The property type is used to define the type of properties. The family type (style) defines a family of systems [77][79]. PADL is an ADL based on process algebra. PADL models architectural style in terms of an intermediate abstraction of architectural type [25]. As discussed in the following section, we can see that the architectural type concept is also the core of ontology-based modeling technology of architectural styles [167][168].

2.1.5 Ontology-based Modeling of Architectural Styles

We have surveyed different approaches and languages, such as ADL-based ACME and process algebra-based PADL, for modeling and analyzing software architectures and architectural styles. Although each of these has different views and capabilities, they all share some common conceptual foundations. Specifically, software architecture and architectural ontology provide a set of common concepts and concerns for describing software architectures. D. Garlan, et al., describe the main elements of component-based architecture and architectural style ontology in [77]:

- Components represent the primary computational elements and data stores of a system.
- Connectors represent interactions among components.
- Systems represent configurations (graphs) of components and connectors.
- Properties represent semantic information about a system and its components that goes beyond structure.
- Constraints represent claims about an architectural design that should remain true even as it evolves over time.
- Styles represent families of related systems. An architectural style typically defines a vocabulary of design element types and rules for composing them [185].

Beyond ADL-based architectural style modeling language, such as ACME, C. Pahl, et al., proposed an ontology-based modeling language for specifying and analyzing architectural styles [167][168]. They defined a generic architectural style ontology based on ontology and description logic [19], which serves as a modeling language with rich and extensible semantic of styles, and operators for combining, comparing and deriving architectural styles and a composition mechanism for incorporating behavioral composition. Their approach mainly includes three parts:

- The basic architectural style ontology

  The ontology based on *ARL* language [19] is defined as

  $$ArchType \subseteq Configuration \bigcup Components \bigcup Connectors \bigcup Role \bigcup Port, \quad (2.2)$$

  and

  $$Configuration \equiv \text{ArchType } \exists\, \text{hasPart}(Components \bigcup Connectors \bigcup Role \bigcup Port), \quad (2.3)$$

  $$Components \equiv \text{ArchType } \exists\, \text{hasInterface}.Port, \quad (2.4)$$

  $$Connectors \equiv \text{ArchType } \exists\, \text{hasEndpoint}.Role, \quad (2.5)$$

  The style ontology consist of five basic elements – *Configuration, Components, Connectors, Role and Port* in (2.2). The hasPart, hasInterface and hasEndpoint are also part of the basic vocabulary.

- Style syntax and semantics for relating architectural styles based on *ARL* [19]

  Based on the elementary type ontology, a style as a specification can be defined as

  $$\text{Style} = \langle \Sigma, \Phi \rangle, \quad (2.6)$$

  where

$$\Sigma = \langle C, R \rangle, \tag{2.7}$$

in which C is a set of concepts and R is a set of roles.

$$\Phi = \{ \phi \mid \phi \text{ is a concept description based on } \Sigma \}. \tag{2.8}$$

Assume that style is interpreted by a set of models M in which the model notion refers to algebraic structures that satisfy all concept description $\phi \in \Phi$. The algebraic structure m in M includes

- Sets of objects C^{a} for each concept c in C
- Relations $R^{I} \subseteq C_i^{I} \times C_j^{I}$ for all roles R: $C_i \rightarrow C_j$

such that m satisfies concept description ϕ.

Based on the style syntax and semantics, the following operators defined in [168] are very useful for style development and analysis:

- Restriction of style $S = \langle \Sigma, \Phi \rangle$ under restriction specification Σ'

  $$\langle \Sigma, \Phi \rangle|_{\Sigma'} \overset{def}{=} \langle \Sigma \cap \Sigma', \{\phi \in \Phi \mid rls(\phi) \in (\Sigma \cap \Sigma') \wedge cpts(\phi) \in cpts(\Sigma \cap \Sigma')\} \rangle, \tag{2.9}$$

  where $rls(\Sigma) = R$ and $cpts(\Sigma) = C$ in which rls and $cpts$ are defined in [168].

- Intersection $S_1 \cap S_2$ of styles $S_1 = \langle \Sigma_1, \Phi_1 \rangle$ and $S_2 = \langle \Sigma_2, \Phi_2 \rangle$

  $$S_1 * S_2 \overset{def}{=} \langle \Sigma_1 \cap \Sigma_2, (\Phi_1 \cup \Phi_2)|_{\Sigma_1 \cap \Sigma_2} \rangle, \tag{2.10}$$

- Union $S_1 \cup S_2$ of styles S_1 and S_2

  $$S_1 + S_2 \overset{def}{=} \langle \Sigma_1 \cup \Sigma_2, \Phi_1 \cup \Phi_2 \rangle, \tag{2.11}$$

- Refinement of style $S = \langle \Sigma, \Phi \rangle$

For any specification $\langle \Sigma', \Phi' \rangle$, if $\Sigma \bigcap \Sigma' = \emptyset$ (Empty), then refinement of S by $\langle \Sigma', \Phi' \rangle$ can be defined as

$$S \oplus \langle \Sigma', \Phi' \rangle \overset{def}{=} \langle \Sigma + \Sigma', \Phi + \Phi' \rangle \qquad (2.12)$$

- Composite elements in architectural styles

  C. Pahl, et al., present the architectural composition principles in [168] based on subsumption which is usually the central relationship in ontology language. The authors introduce the symbol "$\rhd$" to express the composition relationship and define structural composition, sequential composition and behavior composition. For example, (2.3) can be expressed as

  Infrastructure $\rhd$ {*Components, Connectors, Role, Port*}. (2.13)

The ontology-based modeling approach is one of the foundations of my dissertation research on service-oriented enterprise architecture.

2.1.6 Issues and Challenges

This section surveys different approaches in modeling and analyzing software architectures and architectural styles. It also compares formal and informal methods and finds that mixing formal and informal specification of architectural styles is a better way to investigate complex and larger enterprise architectural styles. From the survey, we identify some of the issues and challenges involved in research on software architecture and architectural styles.

- Most formal methods lack the capability to model architectural styles for large, distributed, highly dynamic and complex systems, such as ESOA style and ECSA style systems.

- Given the complexity and dynamism of service-oriented enterprise architecture, especially those based on modern Internet architectures, traditional components-controllers-and-configurations is not enough and not capable for describing and analyzing them.
- Specifying and analyzing dynamism of software architecture and architectural style is one of the challenges.
- Specifying and analyzing quality attributes of software architecture and architectural styles is another challenge under highly dynamic distributed enterprise environments.

2.2 Modeling Enterprise Service-Oriented Architecture

The research work presented in this dissertation is rooted in the following three main research areas: (1) traditional software architecture model and styles, (2) enterprise service-oriented architectural model and its styles which include their formal as well as informal specifications and classifications, and (3) ESOA architectural evaluation.

2.2.1 Traditional EAI Model and Styles

The ESOA is an evolution of traditional Enterprise Application Integration (EAI). The EAI is an integration framework composed of a collection of technologies and services which form a middleware to enable integration of systems and applications across the enterprise.

Erasala, David and Rajkumar [65] conducted a survey of EAI model. EAI software architecture is a middleware-centric integration architecture, such as CORBA, J2EE application servers. Hub-Spoke, EAI Message Broker and Point-to-Point are its major architectural styles.

Giesecke [81] investigates the middleware-induced styles for EAI. He found a way to select middleware for EAI architecture based on different styles – CORBA style, RPC style, ETL style and MOM style.

Andersson and Johnson [14] describe architectural integration styles for large-scale enterprise software systems. These styles include

- Database gateway style
- Desktop integration style
- Message route style
- Database federation style
- Point to point style
- Adapter style

Hohpe and Woolf [94] define and describe 65 enterprise integration patterns for EAI. Although EAI is moving to ESOA in enterprise, most of the patterns they defined will be used in ESOA architecture analysis and design.

2.2.2 SOA Model and Specification

The triangle model (Figure 2.1) of SOA presented in the literature is a basic SOA model [140]. It presents the interaction model of three parties – Service Provider, Service Broker and service Requester in SOA. However, it does not provide specific features in SOA or ESOA, for example, SOA Quality and SOA management.

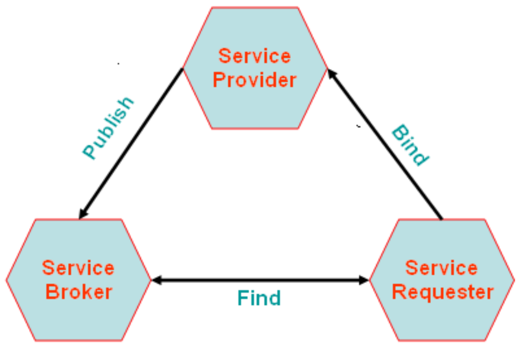


Figure 2.1. Triangle Model of SOA

OASIS [154] develops and proposes an SOA Reference Model (SOA-RM) which defines SOA as a paradigm for organizing and utilizing distributed capabilities that may be under the control of different ownership domains. It provides a uniform means to offer, discover, interact with and use capabilities to produce desired effects consistent with measurable preconditions and expectations. The SOA-RM specification bases its definition of SOA around the concept of "needs and capabilities", where SOA provides a mechanism for matching needs of service consumers with capabilities provided by service providers.

Some of the research work of SOA modeling and specification is based on UML2 profile. Butler [34] proposes an SOA metamodel (SAE) based on UML 2.0 profile. The approach provides a vehicle to quickly begin using the existing UML tools for visually depicting SOA. OMG [161] released its Service-Oriented Architecture Modeling Language (SoaML) on 08/04/2008. The SoaML is also based on UML2 profile, which provides a tool for modeling services behavior through UML2 collaborations platform-independently.

Many SOA industrial players, such as IBM [91], Oracle [163], SUN [195], SAP [214] and Microsoft [141], develop their specific SOA and ESOA models which tie to their products and implementation.

2.2.3 Modeling and Validating SOA vs. Style

Lublinky [127] defines SOA as an architectural style which is similar to the definition of enterprise SOA as an architectural style in my research work on ESOA [201][202][203][204]. He proposed an enterprise SOA conceptual model in which SOA consists of its elements – organization, business model, business process, semantic data model, documents, services and information. My research work defines ESOA style through a 7-tuple model which is based on the domain ontology model of service-oriented enterprise [204].

Baresi, Hecke, Thone and Varro proposed several UML models for modeling, refinement and validation of SOA style applications in [23]. The approach is based on graphs and graph transformation. Based on a formal interpretation of the approach, the consistency between SOA platform and applications can be validated.

2.2.4 SOA Architecture Evaluation

Brien, Bass and Merson [158][159] give a detailed analysis of SOA quality attributes. Their work addresses the relationship between SOA and software architectural quality attributes. They discuss the thirteen quality attributes which should be taken into consideration when designing an SOA application system. It evaluates how quality attributes guide SOA system design and what is the impact of an SOA approach on the quality attributes. The quality attributes become the basis for the SOA architecture evaluation.

Bianco, Kotermanski and Merson [27] propose an evaluation method for evaluating SOA based on quality attributes. They list important SOA design questions that affect quality attributes and discuss each quality attribute through evaluation questions.

Choi, Her and Kim [42] define seven important SOA quality attributes (QA) – availability, performance, reliability, usability, discoverability, adaptability and composability and define metrics for each QA. Their model analysis and SOA evaluation is based on SOA consumers' prospective as the first requirement.

2.2.5 SOA Application Architectures and Styles Classification

Tsai, et al., [214] develop a new methodology and model for classifying SOA-based application architectures that have been proposed based on the structure of application, the runtime re-composition capability, the fault-tolerance capability, and the system engineering support.

Zhao, Dong, and Peng develop classification [248] on ontology and semantic web technologies for software development.

Tang, Dong, Zhao and Tsai investigate a classification of ESOA styles in [203][204]. The five major ESOA styles are compared based on the ESOA model in [201][202][204].

Cesare, et al., compare RESTful web services with traditional web services in architectural principles and decision in [37]. Their work is close to the comparison between EWOA style and EWS-* style [203][204]. However, the comparison in [203][204] emphasizes architectural styles – their common parts and constraints.

2.2.6 SOA Design Patterns

Pahl and Barrett [166] present a modeling and transformation method for service-based software systems. Architectural configurations, expressed through architectural patterns, form the core of an underlying specification and transformation calculus. Patterns at different levels of abstraction form transformation invariants that structure and constrain the transformation process. They explore the role that layered patterns can play in modeling and as invariants for transformation techniques.

Erl [68] provides a set of important SOA design patterns in the book. The SOA patterns are categorized into three groups – Service Inventory Design Patterns, Service Design Patterns, and Service Composition Design Patterns.

2.2.7 SOA Formalization

Various formal models [1][32][58][84][200] for services and service processes have been developed. Table 2.3 summarizes the formal methods for SOA modeling:

Table 2.3. SOA Formalization

SOA & Standard	**Formal Methods & Formalism**
General Modeling	• SO-SAM: PN and TL [73] • SCC: π - calculus, Orc logic [29] • ArchiMate: π -methods[118]
Specification: WSDL	• Z notation [232]
orchestration WS-BPEL	• YAWL: Peril Net [1] • DySco: CSP [171] • Orc: CCS, CSP and Kleene Algebra [144] • Web π_∞ Calculus: timed extension of π -calculus with a transaction construct [133]
Choreography WS-CDL	• π -Process: π - calculus [58] • SCIFF: ALP, CLP [5] • GMF: π -calculus and Solos calculus [102][103]
Design	• ASDL: ITL and CSP [193]

However, the formal model for ESOA and ECSA has not been defined.

2.2.8 Issues and Challenges

Unlike traditional software architectural styles, SOA and ESOA are relatively new architectural styles. Most of the research surveyed in the section either focused on some specific aspects or emphasized their structure based on the vendor's architectural approach. Therefore, the whole architectural style model of ESOA needs to be built for better understanding of complex enterprise SOA systems for both researchers and practitioners. Some researchers view SOA and ESOA as architectural styles, but the formal description and analysis of architectural quality attributes of ESOA have not been investigated. [204] depicts seven characteristics of enterprise architectures as shown in Table 2.4:

Table 2.4. Characteristics of Enterprise Architectures

Characteristics	Description
Customer oriented	• System design is based on customers' requirements • System is interacted by customers • Systems serve customers and meet business demands
Heterogeneous environment	• Complex organization structure • Multiple-vendor software products
Distributed computing	• Everything is connected by enterprise networks • Software systems are highly distributed through the Internet
Integration	• Modern software systems are integrated with existing legacy systems • Different systems in different organizations or different enterprises are integrated.
Manage and access customer/business data and system data	• Business and customer data are managed and can be accessed • System data are managed and can be accessed
Support business processes	• Business transactions • Workflows
Meet certain non-functional requirements	• Performance • Security • Scalability • Agility • Flexibility • Extensibility • Reusability • On time and within budget • Other requirements

Therefore, building a framework for modeling and analyzing enterprise SOA is very challenging.

2.3 Modeling Enterprise Cloud Service Architecture

Current work on bridging ESOA and Cloud Computing (CC) can be classified in the following categories: (1) Specifying and analyzing cloud service-oriented architectural style or framework; (2) CC and SOA convergence in enterprises; (3) creating new approaches combining

SOA approaches and cloud computing which includes bringing SOA best practices into cloud computing and adopting cloud computing power for improving existing ESOA architectures.

2.3.1 Specifying and analyzing cloud service architecture

Zhang and Zhou [243] proposed a Cloud Computing Open Architecture (CCOA). The CCOA is a cloud computing service-oriented architecture framework which bridges the power of SOA and virtualization in the context of Cloud Computing ecosystems. Seven principles of cloud computing architecture are also presented in [243].

Many SOA software venders, such as IBM [98], HP [95], SUN [197], Oracle [164] and Microsoft [142], proposed their new software product architectural models and frameworks which combine the SOA and cloud computing powers. The model presented in [243] can be used for evaluating the architectures of the different approaches.

2.3.2 Cloud Computing and SOA convergence in enterprises

Linthicum presents the dream team of cloud computing and SOA in [125]. He points out that SOA and cloud computing provide a great deal of value when they work together. His book describes the relationship of SOA and cloud computing and guides enterprises on how to make cloud computing and SOA convergence step-by-step. We describe the relationship of SOA and cloud computing through the hybrid architectural style.

Lakshman [115] shows how cloud stretches the SOA scope and proposes a process for idendifying cloud scenarios. Its case studies show that the Microsoft cloud Azure integrates into the enterprise SOA system.

Lawson [119] points out that SOA can help cloud computing in three aspects: (1) One can move services around as needed – including to a cloud server – to address pressing business needs. (2) One can take advantage of virtualization or, as Linthicum explains, address “core applications as logical instances that may run on any number of physical server instances, providing better resource utilization and scalability.” (3) One can create mashups or on-the-fly composite applications with services and leverage the cloud's computing power.

2.3.3 Cloud Architecture

Varia [222] introduces the cloud architectures of Amazon Web Services (AWS). AWS is a typical example of adopting ESOA’s web services (SOAP or REST) to their cloud architectures. DeCandia, et al., [57] present Amazon service-oriented cloud architecture through Dynamo design and how SOA governance can help clouds in achieving high performance and availability.

Dornemann [64] proposes an approach that extends an open source SOA BPEL implementation to use Amazon’s EC2 for providing the process with dynamic resources.

Several research works, such as [173][222][236], describe Amazon cloud, Google cloud, and Saleforce cloud.

2.3.4 Issues and Challenges

If SOA and ESOA are in their early stages, cloud computing is still in its infancy stage. From current research works, we can see that the research and practice of cloud computing tends to combine cloud computing with service-oriented computing. The existing technology for modeling and analyzing enterprise cloud service architecture is still immature and cloud computing is very complicated as well as dynamic with many issues, such as performance, security, and availability. Hence, building a framework for modeling and analyzing enterprise cloud service architecture is very challenging.

2.4 SLA-Aware Enterprise Service Computing

A body of research exists related to our work, which can be categorized as follows: (1) SLA standards and languages; (2) Modeling SLA and QoS; (3) SLA-Aware SOI; (4) SLA Management and SLM; and (5) Adaptive and Automated Computing.

2.4.1 SLA Frameworks, Standards and Languages

There are several SLA frameworks, standards, and languages for SOA systems based on web services. This section introduces SLA frameworks, standards, and languages as well as some other related research works.

The Web Service Level Agreement (WSLA) [55][105][128] is a specification and reference implementation proposed by IBM. The WSLA provides a framework for specifying and monitoring SLA for web services, which includes:

- A Runtime WSLA architecture, and
- An XML-based WSLA language.

The WS-Agreement [128] is a specification from the Open Grid Forum (OGF) which provides an agreement protocol between service consumers and service providers. It uses an extensible XML language for specifying the agreement which includes a negotiation constraint. The specification mainly includes three parts:

- A schema for specifying an agreement;
- A schema for specifying agreement templates to facilitate discovery of compatible agreement parties;
- A set of port types and operations for managing agreement life-cycle which includes creation, expiration and monitoring of agreement states.

The WS-Policy and WS-Policy Attachment [126] are specifications of service qualities which are part of SLA developed by the World Wide Web Consortium (W3C). It is often used in conjunction with other web service specifications, such as WS-Security policy, WS-ReliableMessage policy, and WS-Transaction policy. The specification is not based on agreement but on service quality requirements.

The SLAng [189] is an XML language for defining SLA which is a part of the contracts between web service clients and web services. It has been developed by the TAPAS project at UCL.

The Web Service Offering Language (WSOL) [211][212] is a formal XML language compatible with the Web Services Description Language (WSDL). While WSDL is used for describing operations provided by web services, WSOL provides a formal specification of multiple classes of services for one web service. The classes of services for a web service are distinguished by different combinations of functional provisions and QoS constraints (non-functional requirements [46]), such as response time, simple access rights, and cost/performance. It allows service consumers to select different classes of services in depth, or based on cost. Hence, it can be used to enable a service provider's provisioning models and a consumer's pay-as-you-go business models.

2.4.2 Modeling and Formalizing SLA and QoS

Modeling and formalizing SLA and QoS has received much attention in the enterprise service computing research community. Traditional SLA is typically specified by plain-text documents, such as Amazon's EC2 Service Level Agreement (http://aws.amazon.com/ec2-sla/). The machine unreadable format would not be usable for QoS management and automated negotiation in today's dynamic and on-demand service computing environments. Enterprise cloud service computing provides a pay-as-you-use business model. Consumers pay for the services and QoS. Without using machine-processable SLA, the service billing system would not be able to automatically calculate charges when users are using the cloud service. Moreover, the service billing system would not be able to automatically reduce the customers' charges when the system fails or exhibits slower performance. Therefore, much research focuses on specifying SLA and QoS as machine readable and processable languages. Moreover, service-oriented enterprises are hard to manage and it is difficult to monitor the quality of their systems in order to satisfy their customers and to reduce the service cost. WSLA [129], WS-Agreement 0, SLAng [189] and WSOL [211], introduced in Section 2.4.1, not only make SLA and QoS machine readable and processable, but also provide formal specifications for system modeling and management. Keller and Ludwig describe a novel WSLA framework for specifying and monitoring SLA for Web services [105]. In addition, Tosic and colleagues developed a management infrastructure to show how WSOL manages web service applications [212].

There is ontology-based SLA and QoS modeling research. Dobson and Sánchez-Macián proposed a unified QoS and SLA ontology [61]. Zhou, et al., developed a DAML-QoS ontology [249] to provide better QoS metric models. They proposed a semantic modeling framework for QoS specification [249]. Zhou and Niemela [250] extended OWL-S by including a QoS specification ontology. In addition, they proposed a novel matchmaking algorithm, which is based on the concept of *QoS profile compatibility*. Kritikos and Plexousakis developed a semantic QoS-based framework for web servive description and discovery using OWL-Q [113].

Rigorous formal modeling is helpful towards reasoning about the structure and behavior of SLA as well as QoS based systems and investigating the issue of the description of SLA. Meng proposed a QCCS [136] formal model to enforce QoS requirements in service composition based on Milner's CCS [200]. Nicola, et al., defined a process calculus for QoS-Aware applications [149]. Chothia and Kleijn introduced Q-Automata [43] for modeling QoS on trust and other quality attributes, such as availability and response time.

2.4.3 SLA-Aware Enterprise Service Computing

SLA-Aware enterprise service computing is receiving attention from many researchers since SLA-Awareness brings forth software quality management and QoS into enterprise service computing and implements the enterprise non-functional requirements. Zeng, et al., proposed a QoS-Aware middleware, Agflow [242], for supporting web service composition based on the QoS model they developed. McGough, et al., defined an end-to-end workflow pipeline – Workflow Management Service (WfMS) [134] which is a real-time QoS aware workflow management system based on both strict and loose QoS guarantees. The guarantee requirements are defined in an XPath document, which is connected to a BPEL engine. Wada, et al., proposed a multiobjective optimization framework $\boldsymbol{E^3}$ for SLA-Aware service composition. SLA-Aware or QoS-Aware approach is also applied to web service selection [126]. The aforementioned work does not include SLA negotiation and dynamic resource scheduling. Brandic, et al., presented novel meta-negotiation architecture for SLA-Aware grid services [30]. Song, et al., proposed a framework which supports resource scheduling in a virtualization environment for achieving QoS [194].

2.4.4 SLA Management and SLM

SLA management and Service Level management (SLM) play important roles in SLA-Aware enterprise service computing. While some research focuses on aspects such as SLA-Aware service composition and workflow, SLA modeling, and specification, there are some research works that emphasize SLA management. This addresses end-to-end scenarios across all layers, including internal and external service interfaces, in an enterprise service computing stack. The SLA@SOI consortium published a series of their research works [190] about SLA-Aware Service Oriented Infrastructure (SOI) empowering the service economy in a flexible and dependable way. Their research works include general as well as multi-level SLA management for SOI [190] and SLA-Aware resource management [51]. The Open group published the SLA Management Handbook [162] from Enterprise Perspective as Volume 4 of a series of SLA management handbooks edited by the TeleManagement FORUM. The book is based on a lot of research and practice in SLA management and aims at a true end-to-end SLA. Yeom, et al., proposed a contract-based web service QoS management system architecture [240]. Badidi, et al., presented a broker-based architecture for web service QoS management (WS-QoSM [21]) which is a QoS-aware web service management architecture based on the common concept of brokerage service to mediate between web service providers and consumers. The management operations are executed by the QoS broker. Bhoj, et al., described SLA management architectures in federated environments which share selective management information across administrative boundaries [26]. The SLM focuses on managing SLA commitments at the service level according to the SLA. Figure 2.2 describes the relationship of Key Quality Indicators (KQI), Key Performance Indicators (KPI), SLA and SLA Monitoring in SLM [162]:

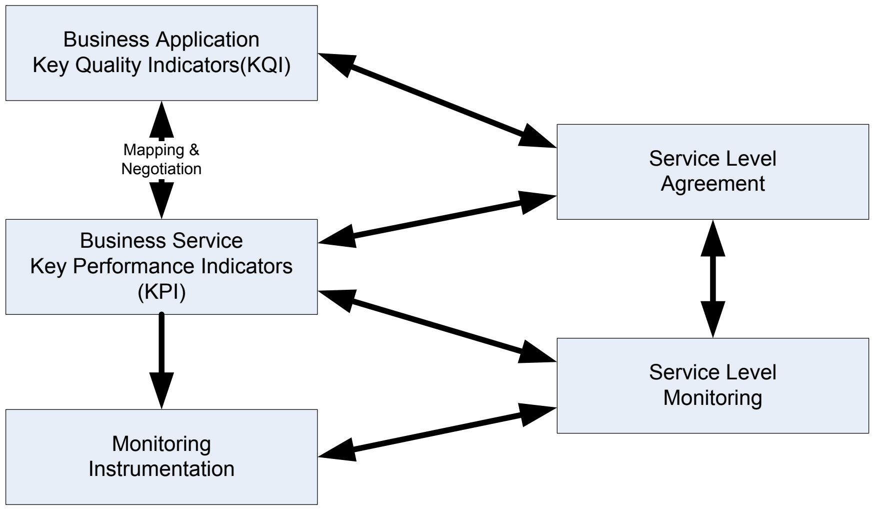


Figure 2.2. Relationship of KQI, KPI, SLA in SLM

Traditional SLM architectures fail to cope with the dynamic runtime nature of enterprise service oriented architecture (ESOA). Schmid and Froeger [186] proposed a decentralized QoS-Management architecture in SOA based on the self-management framework of Service Component Architecture (SCA). Nurmela [150] developed an evaluation framework for SLM in the federated service management context. The SLM not only provides service management for achieving the QoS required by service consumers (enterprise business customers), but also differentiates services [55][80][247]. For instance, a web service can be differentiated into Gold, Silver and Bronze service classes based on KQI and KPI, as defined in the SLO and SLA, with the price of service being associated with each of the service classes. This approach provides a dynamic service provisioning framework and is playing an important role in enterprise cloud service computing.

2.4.5 Adaptive and Automated Computing

SLA-Aware enterprise service computing provides a way to allow enterprises to achieve higher quality assurance and cost-effectiveness in their service oriented architecture systems. However, it also brings forth challenges to distributed service computing in enterprises. The challenges include higher adaptability and automation of enterprise service computing. There is a body of research around the challenges. Yau and An discussed the challenges of adaptive resource allocation for service-based systems [239]. Gao and colleagues presented a QoS analysis technology of adaptive SOA based on a dynamic reconfiguration approach [76]. Wang and colleagues proposed an SLM framework by using QoS monitoring, diagnostics, and adaptation for networked enterprise service oriented systems [233][234]. Self-management [106] and self-adaptive automatic computing [45][76][239] are new challenges for today's SLA-Aware enterprise cloud service computing, such as ECSA [205].

2.4.6 Event-Driven and Real-Time Enterprise Service Computing

Enterprises need automated SLM to make sure they meet SLAs and optimize service delivery in order to improve business outcomes. The SLM for SLA-Aware ESC requires real-time or close to RT visibility, dynamic SLA negotiation, and dynamic system reconfiguration and continuous refinement. However, this level of management is not easy to accomplish with today's distributed and interconnected applications because they execute on heterogeneous systems in different locations. As a result, getting end-to-end visibility to track real-time processes and assure that individual business transactions meet SLAs is a challenging task. Event-Driven Architecture (EDA) [208][204] and RTSOA [220][221] are solutions for this challenge.

2.4.7 Challenges and Research Direction

There are a lot of challenges and opportunities for both researchers and practitioners in the emerging area. [206] summarizes these challenges (or issues) and future research directions as shown in Table 2.5:

Table 2.5. Challenges and research directions

Challenge	**Research Directions**
Theoretical foundation of SLA-Aware enterprise service computing	The SLA-Aware enterprise service computing is a new paradigm of distributed computing. Its theory and formalization is a hot research topic. There are several research directions: • Ontology of SLA-Aware enterprise service computing, such as SLA and QoS ontology [61]. • Formal calculus for programmable QoS, such as Kaos [148]. • Event calculus for WS-Agreement [131].
Modeling SLA-Aware enterprise service computing styles	The SLA-Aware enterprise service computing can be viewed as an architectural style. Modeling the style and its refinement, such as its substyles, is an interesting research topic. Recent trends are • Ontology-based modeling methodology [168]. • Architectural Description Language (ADL) based modeling, such as ACME [78]; Alloy [108]. • Graph-based modeling [23].
Automated and Adaptive SLA-Aware enterprise service computing	The SLA-Aware enterprise service computing requests automated and adaptive service level management automated QoS-pricing computing, SLA-based adaptive optimization, elastic infrastructure and dynamic system reconfiguration. Those requirements introduced many challenging research topics we have outlined in this chapter.
Real-Time or Close to RT SLA-Aware enterprise service computing	To guarantee delivering services and the end-to-end transaction process on SLA in a highly dynamic environment, such as the cloud, SLA-Aware ESC needs to support real-time or close to RT monitoring as well as measurement and management. Event-Driven Architecture (EDA) and RTSOA provide ideas and technologies. How to plug them into SLA-Aware ESC becomes another interesting research direction.
Automated End-to-End and chain SLA in transaction process or workflow	There is a lot of research on SOA process and workflow. However, how to meet SLAs for each service node in the end-to-end transaction process or workflow is a challenge. Modeling the SLA-Aware SOA process and its architecture is worthy of further research.
SLA-Aware application server and enterprise message bus (ESB) and other service process engines	SOA-enabled application servers, such as Weblogic, Websphere, ESB and process engines play an important role – the role of service mediator in enterprise service computing. Researching the next-generation SLA-Aware and adaptive highly-intelligent service mediator is also an exciting project.

2.5 Summary

This chapter presented an in-depth review of the related works and the background of this dissertation. It discusses issues and challenges in research aimed at modeling and analyzing enterprise software architectures and architectural styles. This dissertation builds a framework for resolving some of the issues and accepting some of challenges. This chapter also introduces and analyzes some concepts and methodologies of software architecture and architectural style. These are fundamental basis for the dissertation research. The next five chapters build a framework for modeling and analyzing service-oriented enterprise architectural styles and apply the framework to modeling and analyzing ESOA, ECSA and other enterprise architectural styles.

CHAPTER 3

FRAMEWORK FOR SERVICE-ORIENTED ENTERPRISE ARCHITECTURAL STYLES

We have informally defined enterprise architectural style (EAS) in Chapter 2. EAS is an abstraction of all enterprise software architectural styles, which includes traditional EAS, such as EAI style, and modern service-oriented enterprise architectural styles, such as ESOA and ECSA. The dissertation focuses on modeling and analyzing service-oriented enterprise architectural styles. This chapter presents a framework for modeling enterprise architectural styles using the ontology-based modeling methodology proposed in [168]. The framework includes (1) enterprise architectural style (EAS) ontology, (2) EAS style syntax and semantics, and (3) definitions of ESOA and ECSA.

3.1 EAS Ontology

[168] proposed the basic architectural style ontology (ASO) based on ontology and description logic *ALC* language [19], and it presented an operation calculus for developing architectural styles. [168] also showed how ASO integrates with formal ADLs, such as ACME [79]. However, the basic vocabulary contained in the ontology consists of five elements – Configuration, Component, Connector, Port, and Role – which are suitable for modeling traditional component-based architectural styles, such as Client-Server style, Pipe-Filter style, etc. The five basic elements are not enough to describe enterprise architectural styles, such as ESOA and ECC,

since (1) they are lacking of concept and description of enterprise infrastructure, management, and process; (2) system family in traditional styles lived on dedicated physical servers with static configuration in certain period of time; however, modern enterprise systems in ESOA or ECSA styles are highly integrated, distributed, and dynamic. Specifically, for virtualized infrastructure in ECC, physical resources, such as network and servers are dynamically connected and reconnected on-demand. The virtual workloads are mobilized and look like fluid – changing dynamically. Therefore, it is hard to model virtualized infrastructures and elastic resource management (in which "elastic" means allocating computer resources dynamically.) using traditional style models. We extend the basic architectural style ontology to EAS ontology by using nine parts – Infrastructure, Management (or Governance), Process, Configuration, Component, Connector, Port, Role, and Quality Attributes (QA) and we introduce the time *t* as a parameter in Infrastructure, Management, Process, and Configuration [207]. The time *t* separates each of the four parts into two subparts – static part and dynamic part. If Infrastructure is not changing with time, it is a static Infrastructure; otherwise, it is a dynamic Infrastructure. Similarly, we can define static Management and dynamic Management; static Process and dynamic Process. Each of static Infrastructure, Management, and Process is associated with related static Configuration and each of dynamic Infrastructure, Management, and Process is associated with corresponding dynamic Configuration. The semantics of EAS ontology is defined as follows:

$$\begin{aligned}\text{EAType} \subseteq\ & \text{Infrastructure}(t) \cup \text{Management}\ (t) \cup \text{Process}\ (t) \cup \text{Configuration}\ (t) \cup \\ & \text{Component} \cup \text{Connector} \cup \text{Port} \cup \text{Role} \cup \text{QA}, \qquad (3.1)\end{aligned}$$

in which, EAType denotes enterprise architecture type, and

$$\text{Configuration}\ (t) \equiv \exists \text{hasPart}.(\text{Component} \bigcup \text{Connector} \bigcup \text{Port} \bigcup \text{Role})\ (t), \quad (3.2)$$

$$\text{Component} \equiv \text{EAType} \bigcap \exists \text{hasInterface}.\text{Port}, \quad (3.3)$$

$$\text{Connector} \equiv \text{EAType} \bigcap \exists \text{hasEndpoint}.\text{Role}, \quad (3.4)$$

where Configuration (*t*) represents the application topology; Component is an abstraction of application, component, or service; Connector is an abstraction of the communication (behavior) and glue or link (structure) between Components.

$$\text{Infrastructure}(t) \subseteq \text{IConfiguration}(t) \bigcup \text{IComponent} \bigcup$$
$$\text{IConnector} \bigcup \text{Port} \bigcup \text{Role} \bigcup \text{QA}. \quad (3.5)$$

$$\text{IConfiguration}(t) \equiv \exists \text{hasPart}.(\text{IComponent} \bigcup \text{IConnector} \bigcup \text{Port} \bigcup \text{Role})(t), \quad (3.6)$$

in which t denotes time and IConfiguration(t) is the topology of enterprise infrastructure architecture. If it is invariant with time, then it represents a static topology; if it varies with time, then it denotes a dynamic topology, such as dynamic infrastructure and cloud elastic topology EC2 [9][226].

$$\text{IComponent} \equiv \text{Infrastructure}(t) \bigcap \exists \text{hasInterface}.\text{Port}$$

$$\text{IConnector} \equiv \text{Infrastructure}(t) \bigcap \exists \text{hasEndpoint}.\text{Role}$$

Virtually, the Infrastructure(*t*) can be depicted as shown in the following Infrastracture ontology Figure 3.1:

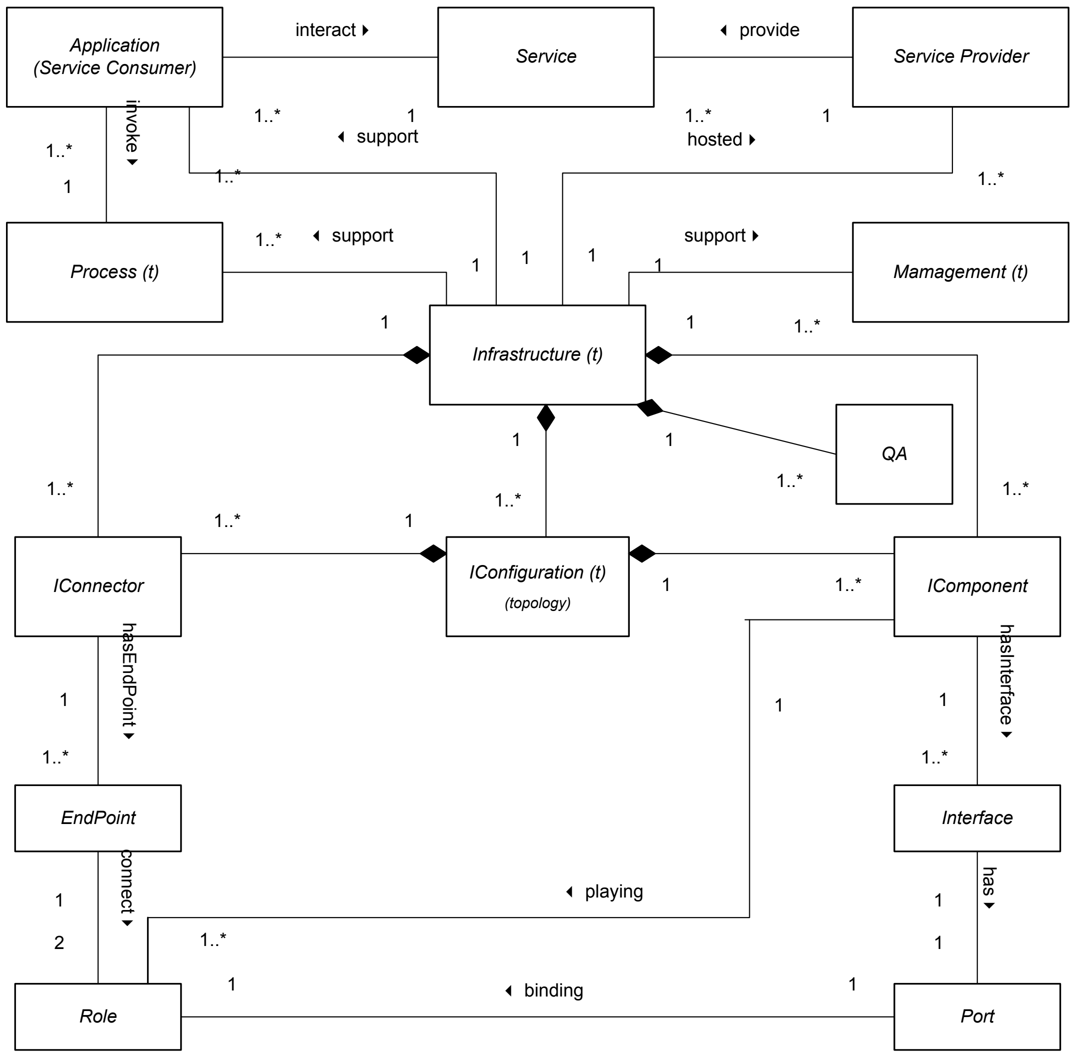


Figure 3.1. Infrastructure Ontology

$$\text{Management}(t) \subseteq \text{MConfiguration}(t) \bigcup \text{MComponent} \bigcup$$

$$\text{MConnector} \bigcup \text{Port} \bigcup \text{Role} \bigcup \text{QA}. \qquad (3.7)$$

$$\text{MConfiguration } (t) \equiv \exists \text{hasPart.}(\text{MComponent} \bigcup \text{MConnector} \bigcup \text{Port} \bigcup \text{Role})(t), \qquad (3.8)$$

in which

$$\text{MComponent} \equiv \text{Management} \bigcap \exists \text{hasInterface.Port}$$

$\text{MConnector} \equiv \text{Management} \sqcap \exists \text{hasEndpoint.Role}$

Similarly, we describe the Management(t) by using a diagram of Management Ontology in Figure 3.2.

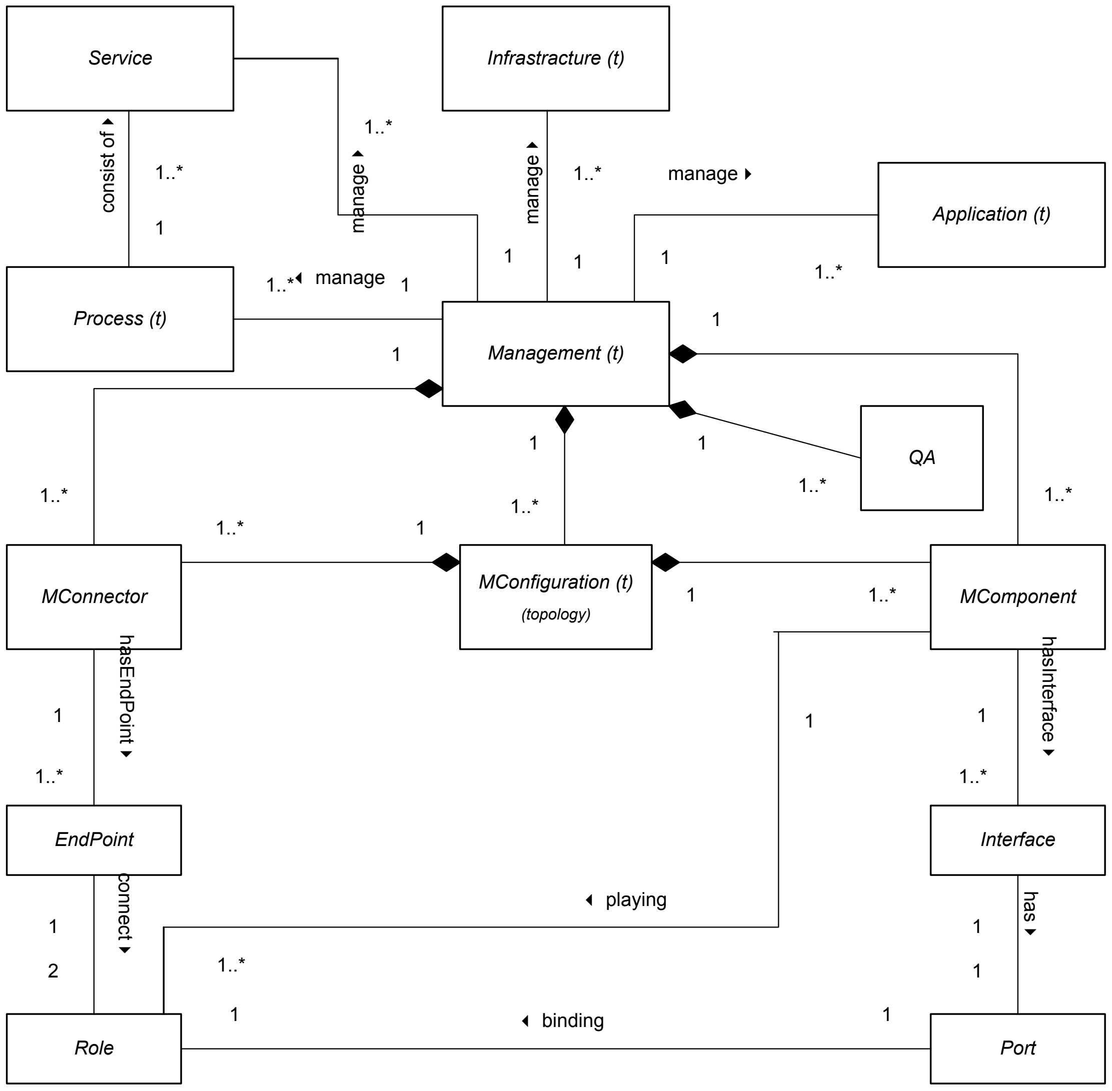


Figure 3.2. Management Ontology

$\text{Process } (t) \subseteq \text{PConfiguration}(t) \cup \text{PComponent} \cup$

$$\text{PConnector} \cup \text{Port} \cup \text{Role} \cup \text{QA}, \qquad (3.9)$$

$$\text{PConfiguration}(t) \equiv \exists \text{hasPart}.(\text{PComponent} \cup \text{PConnector} \cup \text{Port} \cup \text{Role})(t), \qquad (3.10)$$

$$\text{PComponent} \equiv \text{Process} \cap \exists \text{hasInterface}.\text{Port}$$

$$\text{PConnector} \equiv \text{Process} \cap \exists \text{hasEndpoint}.\text{Role}$$

The Process ontology can be described as shown in the following diagram in Figure 3.3:

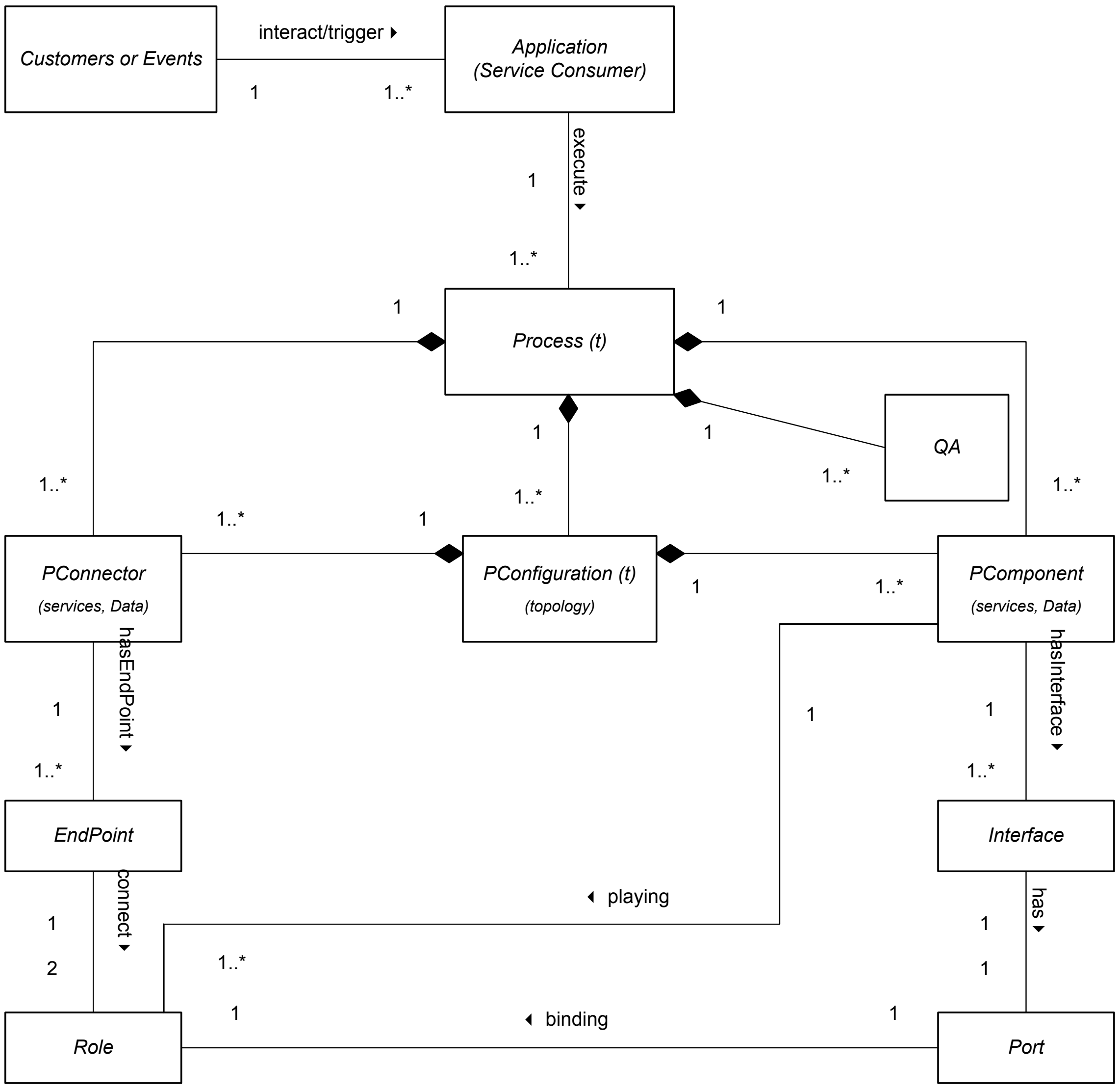


Figure 3.3. Process Ontology

We define quality attributes type QAType as

$$\text{QAType} \subseteq \text{Performance} \bigcup \text{Reliability} \bigcup \text{Scalability} \bigcup \text{Reusability} \bigcup$$
$$\text{Maintainability} \bigcup \text{Security} \bigcup \text{Cost} \bigcup \text{Interoperability} \bigcup$$
$$\text{Availability} \bigcup \text{Flexibility} \bigcup \text{Manageability} \bigcup \text{Agility} \bigcup$$
$$\text{Simplicity} \bigcup \text{Consciousness} \bigcup \text{Accountability} \bigcup \text{OtherQA}. \qquad (3.11)$$

The quality attribute ontology can be defined as:

$$\text{QA} \equiv \exists \text{ hasTradeoff.(Performance} \bigcup \text{Reliability} \bigcup \text{Scalability} \bigcup$$
$$\text{Reusability} \bigcup \text{Maintainability} \bigcup \text{Security} \bigcup \text{Cost} \bigcup$$
$$\text{Interoperability} \bigcup \text{Availability} \bigcup \text{Flexibility} \bigcup$$
$$\text{Manageability} \bigcup \text{Agility} \bigcup \text{Simplicity} \bigcup$$
$$\text{Consciousness} \bigcup \text{Accountability} \bigcup \text{OtherQA}). \qquad (3.12)$$

Here, the roles hasPart, hasInterface, hasEndpoint, and hasTradeoff are part of the basic vocabulary of EA styles. [168] formally defines hasPart by defining an architectural composition principle and introducing a notation $\triangleright$ to express the composition relationship. The composition is syntactically used in the same way as subsumption "$\subseteq$" to related concept description. (3.2) can be formally described as follows [168], for given *t*:

$$\text{Configuration} \triangleright \{\text{Component, Connector, Port, Role}\}, \qquad (3.13)$$

The hasInterface expresses the structural link from components to ports and hasEndpoint denotes the structural links from connectors to roles.

As defined here, the five basic elements - Configuration, Component, Connector, Port, and Role are still the most atomic concepts. The new concepts – Infrastructure, Management, and Process are defined by those basic concepts. They are defined as sub-types of EAType.

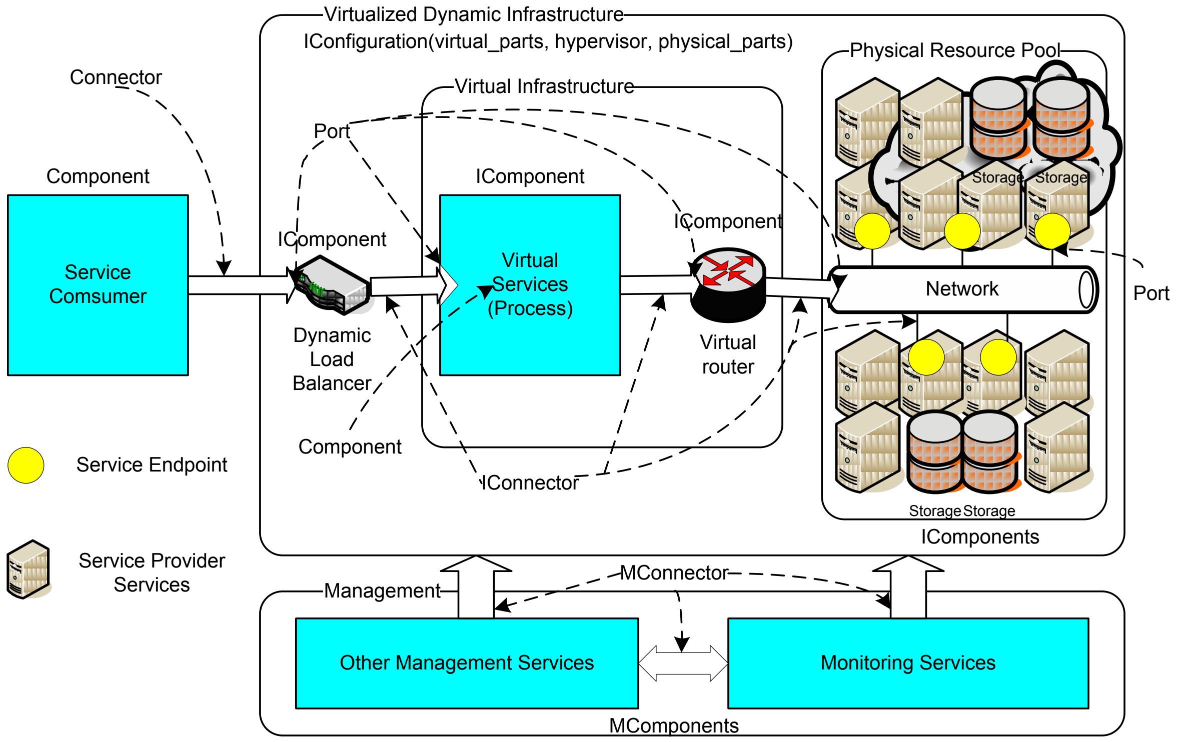


Figure 3.4. Description of Virtualized Enterprise Architecture

Figure 3.4 describes a high level virtualized enterprise architecture and mainly depicts the virtualized enterprise cloud-enabled dynamic Infrastructure in terms of EAS ontology (3.5) and (3.6). Obviously IComponent includes virtual services, virtual servers, virtual network, and physical servers, physical network, physical service providers as well as services in the resource pool. The resources are located in enterprise data center and may be located in cloud providers' data center. IConnector includes LBC-VM, VM-VR, VR-RN, and NR-EP in which LBC = Load

Balancer; VM = Virtual Machine; VR=Virtual Router; RN=Resource Network; EP=End Point of service (Service Port).

The QA in (3.12) can also be formally defined as a composition of architectural quality attributes:

$$\begin{aligned} QA \triangleright \{&\text{Performance, Reliability, Scalability, Reusability, Maintainability,} \\ &\text{Security, Cost, Interoperability, Availability, Flexibility, Manageability,} \\ &\text{Agility, Simplicity, Consciousness, Accountability, OtherQA}\}, \end{aligned} \tag{3.14}$$

The composition is under an optimization constraint through tradeoff [159][207] for consideration of design constraints. The vocabulary we defined is extensible by adding additional parts and elements using the same mechanism based on subsumption and concept description.

3.2 EAS style syntax and semantics

We denote an enterprise architectural style as EAS which is defined by a specification based on the style syntax and semantics as [207]

$$EAS = \langle \Sigma, \Phi \rangle, \tag{3.15}$$

where $\Sigma = \langle C, R \rangle$ consists of concepts C and Roles R. The concept description $\phi \in \Phi$ is based on Σ [19].

To build our framework, we define the following major style operations:

Let $EAS_1 = \langle \Sigma_1, \Phi_1 \rangle$ and $EAS_2 = \langle \Sigma_2, \Phi_2 \rangle$.

- Intersection of *EAS* styles:

  Removing part of concepts and/or roles from one style based on another style is required for style development. The intersection of styles can specify the requirement, which is expressed by $EAS_1 \cap EAS_2$ defined as

  $$EAS_1 * EAS_2 \equiv \left\langle \Sigma_1 \cap \Sigma_2, (\Phi_1 \cup \Phi_2)|_{\Sigma_1 \cap \Sigma_2} \right\rangle. \tag{3.16}$$

- Union of *EAS* styles:

  Adding parts of one style to another style is often required for generating a new hybrid style by two styles. The union of styles is denoted by $EAS_1 \cup EAS_2$ and deals with the scenario which is defined as

  $$EAS_1 + EAS_2 \equiv \langle \Sigma_1 \cup \Sigma_2, \Phi_1 \cup \Phi_2 \rangle. \tag{3.17}$$

- Refinement of *EAS* Styles:

  Refinement of style is important for keeping consistency of all generation of styles, such as a combination of multiple styles, derivation of a new style from an existing style, and extension of a style. Assuming an *EAS* style expressed by (3.15), for any specification $\langle \Sigma', \Phi' \rangle$ with $\Sigma \cap \Sigma' = \varnothing$, the refinement of *EAS* by $\langle \Sigma', \Phi' \rangle$ can be defined as

  $$EAS \oplus \langle \Sigma', \Phi' \rangle \equiv \langle \Sigma + \Sigma', \Phi + \Phi' \rangle. \tag{3.18}$$

  Specifically, we define a notation "$\lhd$" to indicate the style extension relationship. If style EAS_i is an extension of style EAS_j, then

  $$EAS_j \lhd EAS_i, \tag{3.19}$$

3.3 ESOA and ECSA

As we know, ESOA and ECC are different enterprise architectural styles. We have defined the ESOA in [204] and we redefine it based on the ontology-based style notion as

$$ESOA=\langle \Sigma_{ESOA}, \Phi_{ESOA} \rangle, \tag{3.20}$$

where

$$\Sigma_{ESOA}=\langle S, C, D, SI, SM, SP, SQ \rangle. \tag{3.21}$$

In (3.20), *S,C,D,SI,SM,SP,SQ* are defined in Chapter 4, in which *S,C,D* are primary component types in *ESOA, SI, SM, SP* are architectural sub-types – SOA infrastructure type, SOA management type, and SOA process type, respectively. *SQ* is the architecture quality type or attribute which is the set of design constraints. The Φ_{ESOA} in (3.19) can be defined as

$$\Phi_{ESOA}=\{ \phi \mid \phi \text{ is a specification of part in } \Sigma_{ESOA} \}. \tag{3.22}$$

Similarly, we can define

$$ECC=\langle \Sigma_{ECC}, \Phi_{ECC} \rangle, \tag{3.23}$$

where

$$\Sigma_{ECC}=\langle S_c, C_c, D_c, SI_c, SM_c, SP_c, SQ_c, SD_c \rangle, \tag{3.24}$$

$$\Phi_{ECC}=\{ \phi \mid \phi \text{ is a specification of part in } \Sigma_{ECC} \}. \tag{3.25}$$

In (3.24), S_c, C_c, D_c are cloud services, cloud service consumers, and cloud data, respectively, which are primary component types in *ECC*. SI_c, SM_c, SP_c are architectural sub-types and they are cloud service infrastructure, cloud service management, and cloud service process, respectively. SQ_c is the cloud architecture quality type or attribute which is the set of design constraints of cloud systems. SD_c is the cloud specific architecture sub-types which include cloud Development platform, cloud service Deploy type, and cloud service Delivery mode. *ECC* will be discussed in Chapter 5.

3.4 Summary

This chapter presents a framework for modeling and analyzing enterprise architectural styles based on ontology-based modeling technology [167]. The chapter defines nine major vocabularies of enterprise architectural styles and shows how styles can be extended, combined and refined by basic EAS ontology syntax and semantics [207]. The next chapter describes ESOA styles in terms of the framework and research work [201][202][203][204].

CHAPTER 4

ENTERPRISE SERVICE-ORIENTED ARCHITECTURE

This chapter defines and specifies an important enterprise architectural style ESOA based on the framework built in the Chapter 3 and previous research work [201][202][203][204].

4.1 Introducing ESOA

With frequently changing business requirements and the rapid development of technology, enterprise systems have to be built based on adaptable, flexible, and reusable architectures. To reduce coupling, service-oriented architecture (SOA) [44][66][67][96][188][213][215][244] has been applied in many software systems by assembling loosely coupled services that can be used within multiple business domains. SOA provides a flexible set of design principles, constraints, and governing concepts to aid in system design, development, and integration. It defines the interface in terms of protocols and functionality. It also defines service communications by passing data in a shared and well-defined format, or by coordinating an activity among services. SOA can help businesses respond quickly and cost-effectively to changing market-conditions by promoting interoperability, reusability, and extensibility.

Enterprise SOA (ESOA) is a special type of SOA for enterprises. As an architectural style [170][185][70], it is an abstraction of a family of concrete enterprise architectures (instances of a style). It specifies the key aspects of the architectures, and encapsulates important design decisions of common architectural elements and gives emphasis to common constraints as well as their relationships. ESOA combines SOA basic principles and constraints with specific enterprise architecture environment and business requirements (functional and non-functional). From the architectural style prospective, ESOA has more constraints than SOA. The constraints of ESOA are based on enterprise-wide requirements which are specified in Section 2.2.8. ESOA is a new enterprise software architectural style that is an abstraction of concrete SOA architectures in enterprises. Some of the architectures, such as Amazon web service architecture and IBM SOA-based enterprise architecture, are instances of ESOA. ESOA and its substyles focus on service orientation, loose-coupled integration, and interoperability, agility, performance, reliability, reusability, and extensibility. Enterprise systems consist of complex applications in heterogeneous environments. ESOA can better aid application integration because of its interoperability and relatively loose coupling service nature.

Enterprise architecture (EA) is for both businesses and customers. Thus, EA is required to be easy to change with high flexibility. ESOA can help EA to achieve these goals. Moreover, enterprises believe ESOA can enhance their software reusability from “class” to service so that it can help them to reduce their IT costs. Scientists have developed formal service models [32], semiformal service models in UML [23], and formal service interaction models [58]. Recently various ESOA models have been proposed, such as:

- OASIS SOA Reference Model [154] based on the Entity-Relationship (ER) model;
- SOA Meta Model based on the UML profile [34];
- Enterprise Service Bus (ESB) centric ESOA model based on integration notation [38];
- IBM Foundation SOA component model [91];
- Microsoft ESOA model BizTalk [141];

- BEA Aqualogic Service Bus model [56];
- Oracle ESB model [163];
- Pattern-oriented SOA model [241][104]; and
- RESTful model of Web-Oriented Architecture [202].

All these models either focus on the aspects of ESOA which are not abstracted as an architectural style or are vendor-specific implementation of the ESOA. Although a style-based approach for modeling and validating SOA application is proposed in [23], it does not emphasis enterprise-wide SOA application and non-functional constraints.

In addition, lack of understanding of the enterprise service-oriented architectural style and its set of constraints, which includes software architectural quality attributes as well as their tradeoffs, often leads to design-by-buzzword and failure-in-runtime in enterprises. Our research work is motivated by the desire to understand complex ESOA architectural styles and their design constraints with the goal of guiding the ESOA architecture design. My research in this Chapter focuses on ESOA generic model and styles based on our earlier work [62][201][202][204][214][215].

This chapter classifies a generic ESOA architecture model and specifies the ESOA styles. The ESOA model and formal and informal style specification can help with understanding the complex ESOA architectures and in guiding better design of ESOA systems.

This chapter is organized as follows: Section 4.1 discusses the architectural context in service-oriented enterprises Section 4.2 defines a generic ESOA ontology model; Section 4.3 classifies enterprise service-oriented architectural styles into five major styles and defines their hierarchy; Section 4.4 specifies SOAP-based substyle; Section 4.5 specifies REST-based substyle, and the last section compares all the major substyles of ESOA.

4.2 Architectural Context in Service-Oriented Enterprise

This section describes the characteristics of modern enterprise architectures, specifically the architectural context in service-oriented enterprises. Table 2.4 in Section 2.2.8 summarizes various characteristics of modern enterprise information systems.

The characteristics of enterprise information systems are also the basic characteristics of enterprise architecture, which indicate the complexity, requirements, and concerns in designing enterprise architectures. Enterprise Application Integration (EAI) as an integration style provides the principles for building middleware-centric enterprise architectures, such as SUN's J2EE, Microsoft .NET, and has aided in solving more and more complicated enterprise systems integration issues from early 2000 [18]. However, traditional EAI does not fully resolve enterprise integration issues because of the tightly-coupled traditional EAI architecture, lack of good interoperability, and poor scalability as well as security. ESOA, as a better approach than EAI, has been broadly adopted by enterprises since 2003 [135]. ESOA is a new architectural style which is a general SOA style for enterprise architectures.

Figure 4.1 depicts architectural entities and their relationships within a service-oriented enterprise through a domain ontology and reflects the characteristics found in Table 2.4. It is the foundation of our study on generic ESOA architectural model and styles. Compared with other styles, such as client-server and component-based EAI, ESOA styles are service-oriented.

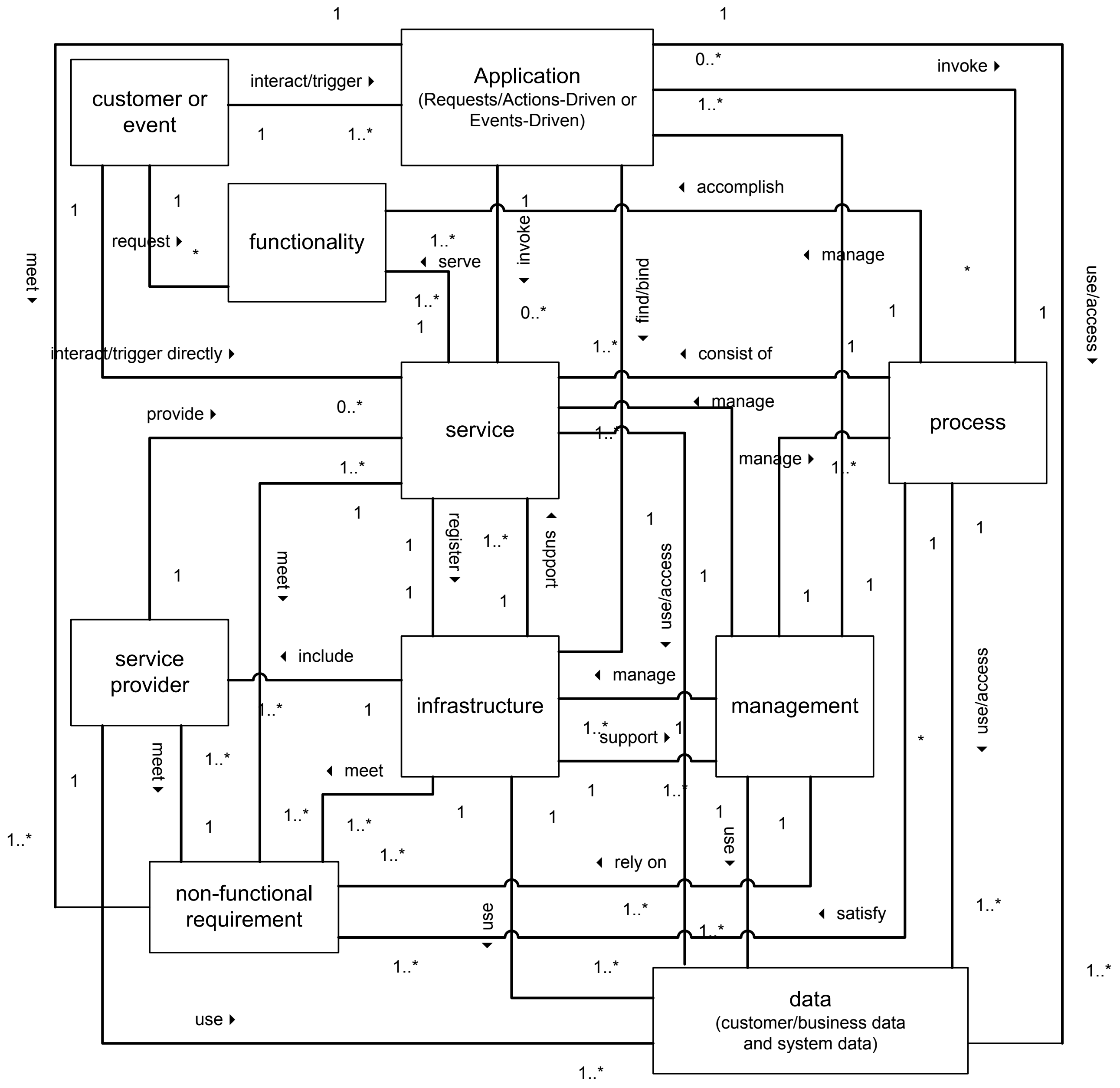


Figure 4.1. Domain Ontology of Service-Oriented Enterprise

The major architectural ontology entities in service-oriented enterprises include:

- Customer or event – A customer is a person who requests business services from the enterprise, and an event is a notable thing ("a significant change in state") that happens inside or outside of an enterprise.
- Application – the interface between customer and service and another kind of service consumer. Application invokes service either by actions or events.
- Service – the operator of the service provider, which is registered in service registry and serves functionalities to customers.
- Service provider – the container or engine of service.
- Process – the coordinated and composed set of services.
- Infrastructure – a set of virtual and physical servers and systems, such as web servers, OS, application servers, database, registry, network, file system.
- Management – the manager and controller of service-oriented systems, such as service life cycle manager, security manager.
- Data – the data includes customer and business data for the enterprise business and system data for defining as well as building a service-oriented architecture and controlling runtime behaviors.
- Functionality – the functional requirement of the system and the service operation served by service.
- Non-functional requirements – the software quality that the service-oriented system should meet.

The domain ontology is independent of any technologies and implementations chosen by enterprises for building a service-oriented system. It not only specifies the relationship among architectural elements, but also describes the relationship between the architecture and customers (business) as well as their requirements. Specifically, the non-functional requirements are specified as constraints for analyzing and designing the service-oriented systems.

4.3 ESOA Ontology

We have defined the ESOA ontology in (3.20) – (3.22). In this section, we give a detailed description of all parts in (3.21). The description of the ESOA ontology in this chapter is based on the domain ontology and in terms of set theory and UML graphic notation. We present (3.20) - (3.22) here as (4.1) - (4.3)

$$ESOA=\langle \Sigma_{ESOA}, \Phi_{ESOA} \rangle, \tag{4.1}$$

where

$$\Sigma_{ESOA}=\langle S, C, D, SI, SM, SP, SQ \rangle, \tag{4.2}$$

$$\Phi_{ESOA}=\{ \phi \mid \phi \text{ is a specification of part in } \Sigma_{ESOA} \}. \tag{4.3}$$

An ESOA ontology is a set of SOA elements, environment, processes, principles, and quality attributes which are specified by the following architectural parts:

$$S=\{ s_i \mid s_i \text{ is a service}\} \tag{4.4}$$

$$C=\{ c_i \mid c_i \text{ is a service consumer}\} \tag{4.5}$$

$$D=\{ d_i \mid d_i \text{ is an SOA data element}\} \tag{4.6}$$

$$SI=\{ r_i \mid r_i \text{ is an SOA infrastructure}\} \tag{4.7}$$

$$SM=\{ m_i \mid m_i \text{ is an SOA management}\} \tag{4.8}$$

$SP = \{ p_i \mid p_i$ is an SOA process$\}$ (4.9)

$SQ = \{ q_i \mid q_i$ is an SOA quality attribute$\}$ (4.10)

Each architectural part is a set of its specific elements in (4.4) to (4.10) in which each element in each set corresponds to one entity of the domain ontology in Figure 4.1. Specifically, the service in *S* corresponds to the "service" entity; the service consumer in *C* is one of the entities, "customer" or "application". The SOA data element in D is the system data part of "data" entity. The SOA infrastructure element in *SI* is the "infrastructure" entity. The SOA management element in *SM* is the "management" entity. The SOA process element in *SP* is the "process" entity. The SOA quality attribute element in *SQ* is the "non-functional requirement" entity. Formula (4.2) can be described by an upper domain ontology diagram as shown in Figure 4.2.

This dissertation differentiates seven different classes of service-oriented architectural parts as ESOA ontology:

- Services;
- Service consumers;
- SOA data elements;
- SOA infrastructure elements;
- SOA management elements;
- SOA processes; and
- Quality attributes.

Services: In formula (4.2), *S* is a finite set of services and a service is the fundamental element of SOA. Informally, a service is a self-contained software abstraction of business, technical functionality, or infrastructure management, defined by a well-defined interface that focuses

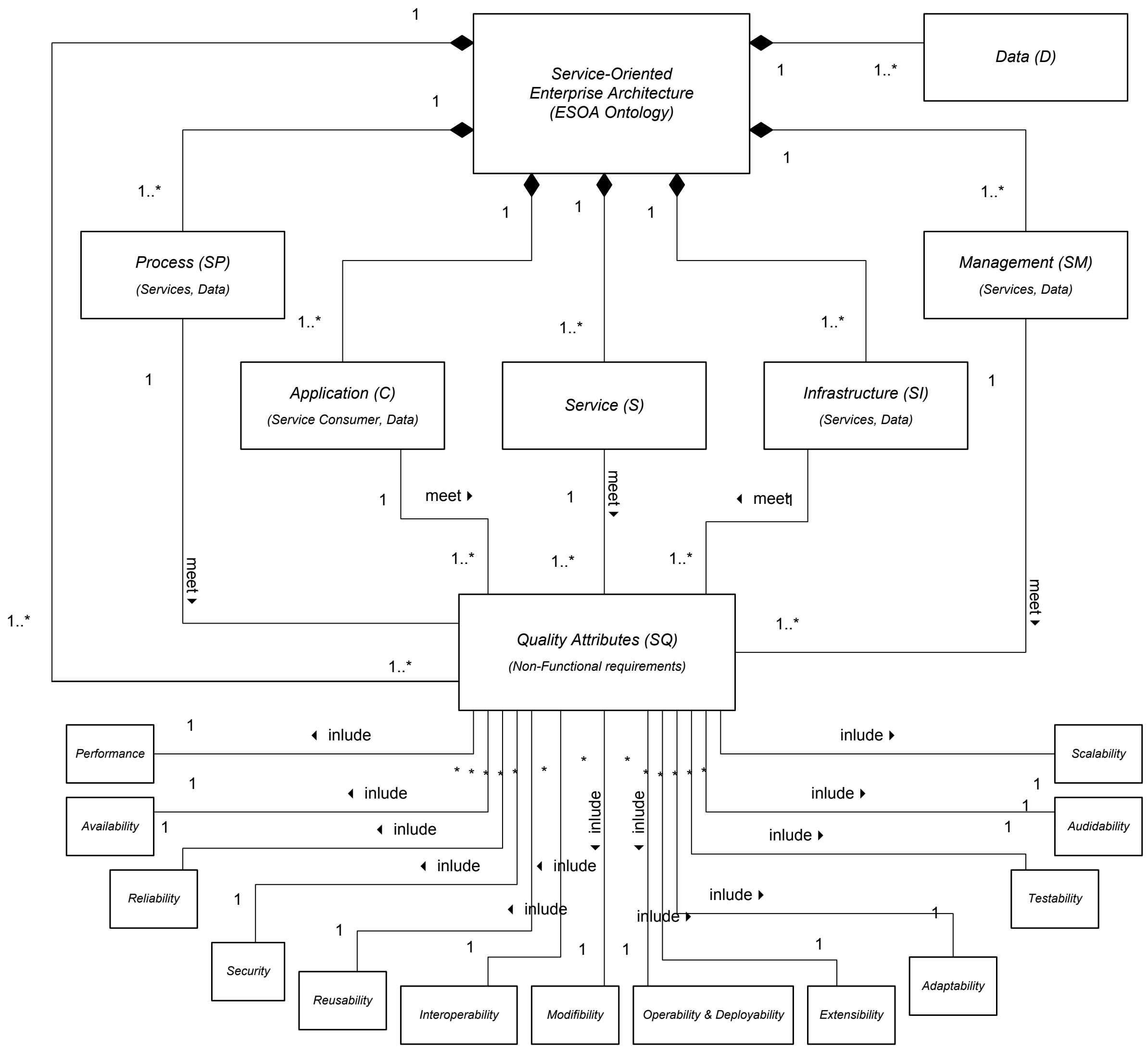

Service-Oriented Enterprise Architecture (ESOA Ontology)
Data (D)
Process (SP) (Services, Data)
Management (SM) (Services, Data)
Application (C) (Service Consumer, Data)
Service (S)
Infrastructure (SI) (Services, Data)
Quality Attributes (SQ) (Non-Functional requirements)
meet
inlude
Performance
Availability
Reliability
Security
Reusability
Interoperability
Modifibility
Operability & Deployability
Extensibility
Adaptability
Testability
Audidability
Scalability
1
1..*
*

Figure 4.2. ESOA Domain Ontology Model

normally on the descriptions of functional aspects, such as input, output, preconditions, and effects known as IOPE [62]. Services in S are published through the service registry in the Service Infrastructure (*SI*). They are found and bound by the facilities in *SI*. In addition, services are consumed by service consumers in *C*. There are three kinds of fundamental services in ESOA in terms of the fundamental service classification in [99]:

- Basic services which are the fundamental elements of ESOA.

- Composed services that are composed from basic services;
- Process services are those services that perform process computations in ESOA.

Service consumers: To serve service consumers in *C* and execute business management processes, composed services and process services are orchestrated and/or choreographed by the Service Process (*SP*). Based on the states of services, services can be classified as stateless services and stateful services. The traditional web services are stateless [67], whereas the Grid services defined by the Web Service Resource Framework (WSRF) [157] are stateful services. Section 4.7 will discuss WSRF services.

SOA Data Elements: These (*D*) include SOA meta-data, policy data, and other service data used by all other parts in the ESOA model. There are two kinds of service representations:

- WSDL-based representation.
- Ontology-based representation, such as OWL-S [228] for describing semantic web services.

The service representation or specification is a subset of *D*.

SOA Infrastructure: This (*SI*) is the heart of ESOA, which discovers, and routes and binds services to proper service providers based on the service requests from a service consumer *C*. In the previous section, *SI* is defined as a layered architecture model. Each layer can consist of a set of services, such as communication service, on-ramp service, and off-ramp service. Thus, any infrastructure service is denoted by $IS \in S \cap SI$.

SOA Management: This (*SM*) controls *SI*, *S,* and *SP*. It relies on SOA quality attributes *SQ*. Four common SOA management functions are provided:

1. The **Business Management** manages the transformation between the business model and the services model: service orchestration and/or service choreography in *SP* for business processes, transaction, and workflow.
2. The **Lifecycle Management** controls *S*, *SI,* and *SP* at service modelling, assembling, service routing, transformation, and versioning.
3. The **Quality-of-Service** (QoS) management provides provisioning and quality of service (QoS) assurance based on the *SQ*. For example QoWS [165] provides QoS management.
4. The **Security and Policy Management** controls service with system level security and policy by using various security definition data in *D* as well as security and policy services in *S*.

SM monitors *S*, *SI,* and *SP* by observing system run-time behaviours, measures various performance and QoS metrics, and reports back to a control agent, such as QoWS.

SOA Processes: This (*SP*) is composed of services in *S* and defined by business management in *SM*. *SP* includes two main kinds of processes [169]:

- Service Orchestration (*SO*) which refers to an executable business process that can interact with both internal and external services.
- Service Choreography (*SC*) which defines the interaction between independently defined processes.

They can be formalized by Petri nets [137], π-calculus [58], or other formalisms. Petri nets are suitable for modeling concurrency. The basic elements, e-service net and orchestration net for modeling the *SO*, are developed in [137]. IBM's Web Services Flow Language (WSFL) adopts Petri nets for expressing the service process logic. The π-calculus is a kind of process algebra [143] for modeling processes. It can be used for modeling *SO* and *SC* [58]. Moreover, the SOA process languages, such as Microsoft's XLANG and Oracle's WS-CDL, are inspired by π-calculus. Note that once formalized, software services can be analyzed using numerous tools that have already been developed. For example, if a service has been formalized by Petri nets, reachability and deadlock analyses can be conducted.

SOA quality attributes: These (*SQ*) are important to all other parts for architectural decisions and design. The quality attributes are constraints for structure and behavior of services, processes, infrastructures, and management. They provide the principles and guidelines for analyzing and designing ESOA. For instance, the extended service defined in (4.4) can be called a "Governed service" depicted in Figure 4.3

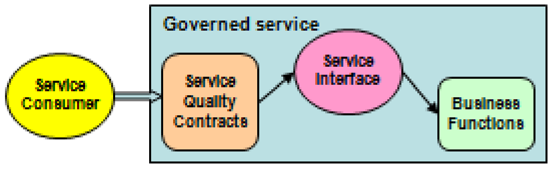


Figure 4.3. Governed Service View

Ontology model (4.1) – (4.3) is an abstraction of general ESOA architectures which include different families of ESOA architectures, such as web service SOA architectures based on SOAP (Simple Object Access Protocol) [66] and Representational State Transfer (REST) web service SOA architectures based on HTTP protocol [70]. For different ESOA architectural families, the above seven parts can be specified with their different characteristics.

4.4 Enterprise Service-Oriented Architectural Styles

Section 4.3 defined ESOA ontology. There are different families of ESOA architectures. In general, an architectural style defines a family of architectures with common structure and constraints. The enterprise service-oriented architectural styles are abstractions from different families of ESOA architectures. Like SOA architectural style, the ESOA architectural style is the umbrella of all different ESOA substyles. Figure 4.4 shows various ESOA architectural styles based on ESOA ontology defined in Section 4.3:

- **Services**: they are the building blocks in any ESOA systems.
- **Service consumers**: they are customers or customer facing applications of enterprise business. Services shall provide business and technical services for consumers.
- **Data elements**: Various representations and data will be used to specify services, workflows, and data used in ESOA.
- **Infrastructure**: Enterprise services and providers must be supported by ESOA infrastructure for guaranteeing QoS.
- **Management**: Enterprise services must be managed and controlled by SOA management services based on Service Level Agreement (SLA).
- **Processes**: Enterprise services shall be capable of executing business processes and workflows.
- **Quality attributes**: Enterprise services and their supports as well as management shall satisfy both functional and non-functional requirements.

The SOAP-based enterprise service architecture is the first substyle of ESOA, called EWS-* style in this chapter.

The Web SOA (WSOA) based on REST architectural style [70][74] and enterprise Web 2.0 is another substyle (called EWOA) [202] of the ESOA.

Unlike the request-driven styles, such as EWS-* style and EWOA style, the Enterprise Event-Driven SOA (EEDA) is an event-driven style [178][208].

Because of the maturity of component-based technology and application server, the enterprise component-based services architecture (ECBS), such as the Service Component Architecture (SCA) [153] and J2EE component-based enterprise services approach [33], is another substyle of ESOA.

Unlike the above styles, the enterprise grid-enabled SOA, called EGSA style in this chapter, is a hybrid style of the ESOA style and the grid computing style [36][160][122][191] which coordinates computing resources that are not subject to centralized control and provides dynamic scalability and continuous availability.

In addition, many enterprise systems have used a hybrid approach by combining two or more different ESOA substyles. Figure 4.4 shows the classification of ESOA styles and their hierarchy.

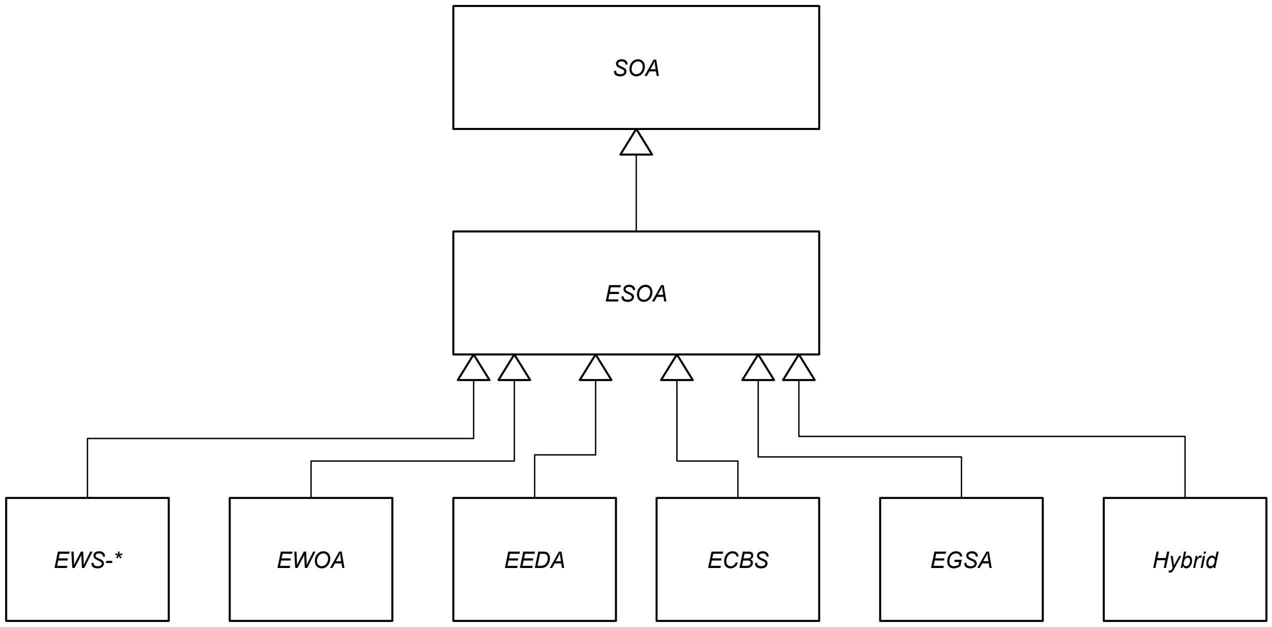


Figure 4.4. ESOA Classification and Hierarchy

Table 4.1 provides the definition and related references for each basic ESOA substyle.

Table 4.1. Description of Basic Substyles

Style	Style keywords	Description
EWS-*	Simple Object Access Protocol (SOAP) Request/Response Web service Web service standards (WS-*)	It is SOAP-based enterprise service architectural style. Specifically the style is based on a series of web service standards called WS-* [235].
EWOA	Representation State Transfer (REST) HTTP protocol Request/Response Web 2.0 Web-Oriented Architecture (WOA)	It is Enterprise Web-Oriented Architectural style. The Web-Oriented Architectural style is first defined by Gartner [74]. The EWOA style is specified in [202].
EEDA	Events Event-Driven Architecture (EDA) Event-Driven Services Complex Events Processing (CEP) Events Channel	It is Enterprise Event-Driven Architectural style which is a hybrid style with ESOA and EDA style. The EDA is introduced in [135] and is defined as a SOA style by Gartner [178].
ECBS	Component-based Service Component Architecture (SCA) Enterprise Java Bean(EJB) Java Business Integration (JBI)	It is Enterprise Component-Based Service Architectural style which is based on service component-based specifications, such as SCA [153] and SUN's JBI [179] as well as EJB [33].
EGSA	Grid Computing Open Grid Services Infrastructure (OGSI) Web Service Resource Framework (WSRF) Grid Standards (OGSI, WS-Resources)	It is Enterprise Grid-Enabled Service Architectural style which is a hybrid style with ESOA and Grid computing style [157].

According to the syntax defined in Section 3.2, we can formally define all substyles as extensions of ESOA as:

ESOA $\lhd$ EWS-*; ESOA $\lhd$ EWOA; ESOA $\lhd$ EEDA;

ESOA $\lhd$ ECBS; ESOA $\lhd$ EGSA.

4.5 Specifying EWS-*

This section presents one of the ESOA substyles EWS-* based on the proposed ESOA ontology model (4.2). The EWS-* defines a family of ESOA architectures – SOAP-based web-services architectures.

4.5.1 Web Service

This dissertation defines an abstract model of services with both functional aspects (operations) and non-functional properties (quality attributes or semantics). Specifically for EWS-* style services, the functional descriptions of services are based on WSDL 2.0 [229] whereas the non-functional descriptions are based on its extensibility that allows extending WSDL 2.0 at both the element and attribute levels. For example, SAWSDEL [111] is the Semantics Annotation for WSDL 2.0, which is the first standard for adding quality semantics into the service descriptions.

A web service is defined as a set of service operations:

$$SO_p = \{ o_i \mid o_i \text{ is an operation}\}, \qquad (4.11)$$

where

$$o_i = \langle n_i, in_i, out_i, \inf_i, outf_i, mcp_i, q_i \rangle$$

in which n_i is the name of operation o_i; in_i is the incoming message of a service and out_i is the outgoing message from a service; $\inf_i$ and $outf_i$ indicates whether a fault (the fault is an event that happens during the execution of a message exchange that disrupts the normal flow of message) is injected into the service or generated by the service, respectively; mcp_i denotes message exchange patterns (WSDL 2.0 supports eight message exchange patterns as shown in Table 4.2); q_i is the set of quality attributes, such as transaction for the operation.

Formally, a web service $s \in S$ is a 5-tuple:

$$s = \langle I, M_s, P_f, l, Q_s \rangle, \tag{4.12}$$

where

$I = \langle SO_p, Q_I \rangle$, which is the service contract or interface in which SO_p is the set of operations provided for service consumers and Q_I is the set of quality attributes for the interface, which can be described by a set of features and a set of properties such as security request features and security-level properties, through WSDL 2.0 extension, such as SAWSDL [111], and SO_p and Q_I define the functional and non-functional behaviors of a service, respectively.

M_s is a set of internal states of the service, which can represent any information it manages, such as variables, service lifecycle states [231], and interaction states.

P_f is the internal process which denotes the service functionality encoded by the formalism f. P_f denotes a service implementation model which is not visible outside of the service. The functionalities implemented by P_f provide services to consumers through the interface.

l defines the service location, such as a set of endpoints: l = $\{ep \mid ep$ is an endpoint$\}$ for SOAP-based web services. An endpoint indicates an association between a binding and a network address, specified by a URI, that may be used to communicate with an instance of a service. Formally, for a web service,

$$ep = \langle Binding, Address \rangle$$

Table 4.2. Message Exchange Patterns

Pattern name	**Description**
In-Only	A standard ***one-way*** message exchange where the consumer sends a message to the service provider only.
Robust In-Only	This pattern is for reliable one-way message exchanges. The consumer (client) initiates a message to which the service provider responds with status. If the response is a status, the exchange is complete. If the response is a fault, the consumer must respond with a status.
In-Out	This is equivalent to ***request-response***. A standard two-way message exchange where the consumer (client) sends a message, the service provider responds with a message or fault and the consumer responds with a status.
In Optional-Out	A standard two-way message exchange where the provider's response is optional.
Out-Only	The service operation produces an out-only message, and cannot trigger a fault. This is equivalent to the ***notification*** message pattern.
Robust Out-Only	The service operation produces an out-only message, and can trigger a fault.
Out-In	This is equivalent to the ***solicit-response*** message pattern. The service sends a message to the consumer and receives a response message from the consumer.
Out Optional-In	The service produces an out message first, which may optionally be followed by an inbound response.

where the *Binding* specifies a concrete message format and transmission protocol used for defining the endpoint; the *Address* is an optional WS-Addressing reference.

$Q_s \subseteq SQ$, which defines service quality attributes such as interoperability, performance, and security at the service level, also called Quality of Service (QoS) and Service Level Agreement (SLA), as well as service properties S_p. We define

$$Q_s = \{\text{common quality attributes}\} \cup S_p \tag{4.13}$$

where the common quality attributes for SOA are described in [158] and

S_p = {standardized service contracts, reusability, relative autonomy,
statelessness, discoverability, relative loose coupling,
abstraction, composability}

Q_s is also called the set of non-functional properties and can be described through service ontology of Web Service Modeling Ontology (WSMO) [224] or Ontology Semantic Markup for Web Services (OWL-S) [132] and linked through the extensibility of WSDL 2.0.

Formula (4.12) is an extension of the formula of service in [84]. The 5-tuple is an abstraction of service in EWS-*, which abstracts away from the concrete service implemented by a specific technology with particular formalism f and the representation of the internal states. Formula (4.12) defines both service structures and its behaviors. I defines both functional and non-functional contracts which include data and service behaviors. M_s,P_f, Q_s mainly define the internal service logic which is the implementation of the contracts of data and behaviors defined by I. | defines an endpoint where a service can be accessed. Visually, a web service can be depicted as shown in Figure 4.5:

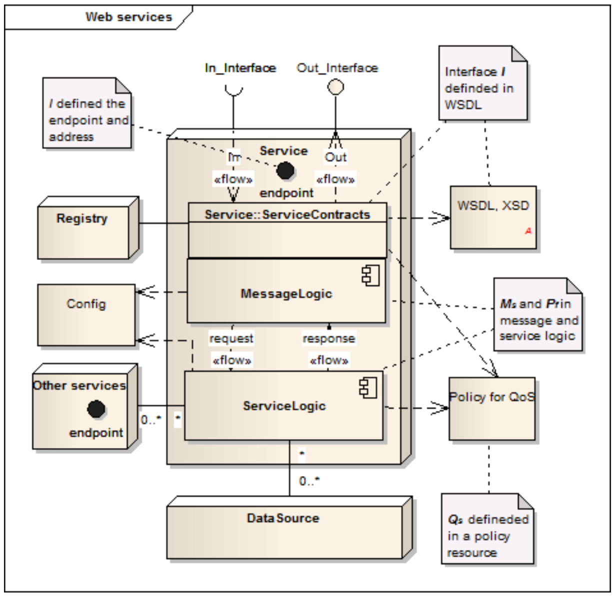


Figure 4.5. UML Model of Web Service

Figure 4.5 not only describes formula (4.12) and formula (4.13) visually, but also depicts the relationship between the service and external system, such as the Data Sources and the Registry, which belong to *SI* in (4.2). Moreover, Figure 4.5 shows the service composability which means services can be composable for completing a business task, such as a transaction. It is one of the service properties in S_p and is a pre-condition for constructing *SP*.

4.5.2 Service Consumers

The traditional service consumers are any applications which can access web services, such as a web-based J2EE application or a .NET application. An action-based service consumer is defined as a 5-tuple

$$c = \langle E_c, F_c, A_c, M_c, Q_c \rangle, \tag{4.14}$$

where

E_c is a set of elements including

- Data elements, such as data objects;

- Component elements, such as controller, filter, state manager, web cache;
- Connector elements, such as adaptor, AJAX;

F_c is a set of forms which includes

- User interfaces;
- Web-based interfaces;
- System properties;

A_c is a set of actions which includes

- External actions, such as an event trigger for service request or a SOAP message sending to a SOAP-based web service;
- Internal actions, such as a trigger for operations or a reply processed by the services;

M_c is a set of internal states that determine the consumer's behaviour.

$Q_c \subseteq SQ$ is a set of client system quality attributes.

A typical action-based web service consumer can be modeled as shown in Figure 4.6 whose components can be mapped to the formula (4.14) as follows:

E_c = {Connectors, Controller, Filters, Domain objects, State manager,

Business delegation objects, Widgets},

F_c = {Web forms, GUI components, Interface Config, Server config},

A_c = {Action Events, Action Handlers},

M_c = {GUI States, States},

Q_c = {security (through SSO, ACL), performance (defined in Config)}.

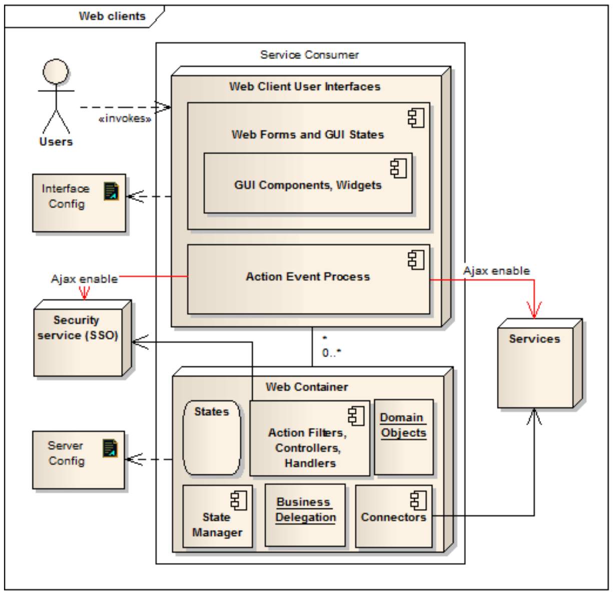


Figure 4.6. Action-Based Service Consumer

Once this aspect is formally specified, one can perform various analyses based on service consumers:

1) **User-service interaction**: The interaction can be short and brief like UML use cases, or can be extensive like use case scenarios [216] and anything in between [121][123]. The user-service interaction can be useful for system composition, integration testing, and automated test script generation. For example, given the user-service interaction specification of a consumer, it is easy to verify that a given service can provide the needed service.

2) **Profiling and provisioning**: Once the workflow and usage pattern of a service consumer are known, the information can be useful for profiling and resource provisioning. This is important for QoS-based system evaluation and assurance.

3) **GUI representation:** System GUI can be formally specified and analyzed [217] for completeness analysis (all inputs or combinations of inputs can be handled by the system, and each input button in the GUI has the corresponding action routine to respond),

reachability analysis (all paths lead to an end state), and consistency analysis (the system will deliver consistent answer for consistent input).

4) **Data flow analysis:** From input-output and from an architecture description of interconnecting services, one can show the flow from one service to another service. This information can be useful for data provenance and various data analysis such as data integrity analysis.

4.5.3 SOA Data

The *D* in (4.2) is a set of SOA data elements that is a finite set. For EWS-* style, the *D* is formed by various web service metadata and data files, such as WSDL files, policy definition files, infrastructure configuration files, resource metadata, and SOA management data. Table 4.3Table 4.3 lists major SOA data elements for EWS-*.

In the SOA data, the service metadata, such as service definition and service registration data, plays a key role in SOA [53].

Note that many data elements are represented by XML and, thus, they can be analyzed or reasoned about based on XML-related tools. For example, many ontology systems, including RDF, OWL, and OWL-S, are all based on XML and many reasoning tools such as DL (Description Logic) can be used for reasoning about them. Policy data can be subjected to completeness and consistency checking, dependency analysis, simulation, and performance evaluation [216][217].

Table 4.3. SOA Data Elements

Data Elements	**Examples of Web Services**
Resource metadata	Metadata of Web services, Clients, Database, mainframes, caches
Resource identifier	UDDI keys, data sources
SOA metadata	XML Schema (XSD) Service description Endpoint schema
Infrastructure data	Administrative metadata
Process data	WS-CDL document BPEL, XLANG documents
Service specification	WSDL documents
Management data	Monitor report, SLA data
Policy data	XACML documents

4.5.4 SOA Infrastructure

The SOA infrastructure is the heart of ESOA, and it supports the transformation of business in an enterprise or between enterprises into a set of managed services *S* or repeatable business tasks, which can be accessed over a network when needed. The network can be a local network, the internet, or a wireless device network. The SOA infrastructure *SI* is the bridge of the transformation between business and services. It is defined as three layers, each of which consists of internal services and components for a traditional ESOA or EWS-* style ESOA:

- Connection layer which includes rich client API, standard protocols, such as HTTP, SOAP, TCP/IP, and adapters. The layer provides connectivity to different application systems and services, such as ERP, CRM, Finance, Shopping/Shipping, and Travel, in different platforms, such as J2EE, .NET, and legacy backend systems.

- Communication layer which includes message services, such as JMS and SOAP Engine, and provides the capability of carrying messages between services as well as transferring various messages, such as XML messages, in a reliable and secure way.
- Mediation layer which includes a set of on-ramps, a set of off-ramps, smart routing services, transformation services of protocols, and data. It provides the semantic glue between disparate services and different applications in enterprises. It includes transport protocol conversion, smart service routing, service invocation and dispatch, etc.

The Enterprise Services Bus (ESB) [38] is one of the implementations of *SI*. It is the core of the SOA infrastructure for a traditional ESOA. For Web SOA, the infrastructure includes the HTTP servers and other web infrastructure. This section focuses on traditional ESOA (EWS-* style).

Another popular trend is the use of a cloud computing environment where service requests are automatically tracked and provisioned like Google's App Engine. Such cloud computing environments are still the subject of active research. A cloud computing infrastructure often consists of three separate infrastructures: Infrastructure-as-a-Service (IaaS) [125], Platform-as-a-Service (PaaS) [125], and Software-as-a-Service (SaaS) [125] . IaaS is at the bottom and SaaS is at the top with respect to user interaction. The dissertation will discuss and model the new enterprise architecture style in Chapter 6.

4.5.5 SOA Management

The SOA Management *SM* plays an important role in ESOA. It is defined as a 5-tuple for an EWS-* style

$$SM = (I_m, C_m, S_m, A_m, Q_m), \tag{4.15}$$

where

I_m is a set of SOA management interfaces which provide management operations to the EWS-* system;

C_m is a channel of SOA management which provides the connectivity and communication between the management interfaces and service interfaces;

S_m is a set of management servers which include the directory server, messaging server, policy server, and service management server. They provide a set of management operations:

- Resource management: Services as system resources;
- Service and infrastructure discovery;
- Network and application monitoring;
- Policy enforcement;
- Service-level agreement management;
- Exception management;
- Closed-loop governance; and
- Service lifecycle management.

A_m is a set of distributed agents which monitor any ESOA-style system;

$Q_m \subset SQ \cup D$, which is a set of metadata and quality attributes specified in management policies and service-level agreement.

Figure 4.7 describes the relationship among elements in (4.15).

For an EWS-* style, OASIS has proposed several standards: WSDM (Web Service Distributed Management) [156], MUWS (Web Service Using Management) [152] and MOWS

(Management of Web Services) [151] for web services SOA management. For an EWS-* style with WSDM, the Management Channel C_m is a part of the SOA infrastructure, the Management Interface I_m is the set of web services endpoints, the Management Service S_m is the set of web services for management, the distributed agent A_m is a set of management agents, and the quality Q_m is a set of quality attributes and properties of services and its infrastructures.

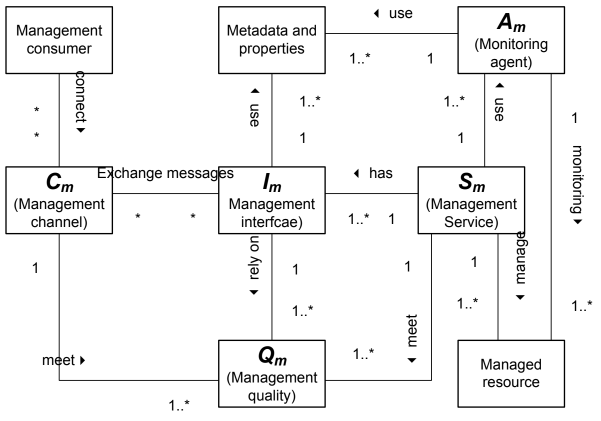


Figure 4.7. SOA Management

4.5.6 SOA Process

The SOA Process *SP* is an important part of any ESOA-style system since a complex business task must be completed in multiple steps of business processes. The business processes can be executed by multiple services which are managed in an SOA process. An SOA process includes a set of composite and/or coordinated services in various process patterns, such as sequence or parallel. Its major elements [169] are shown in Table 4.4.

Figure 4.8 shows the general SOA processes. Different ESOA styles have different SOA process styles. An EWS-* process is mainly based on two WS-* standards:

- Web Services Business Process Execution Language (WS-BPEL)
- Web Services Choreography Description Language (WS-CDL)

Table 4.4. SOA Processes

Service Process	**Examples of Web Service Process Standards**
Orchestration	Web services Composition: WS-BPEL, XPATH
Choreography	Web services coordination: WS-CDL, BPML,WSC I

Figure 4.8 shows the general SOA processes. Different ESOA styles have different SOA process styles. An EWS-* process is mainly based on two WS-* standards:

- Web Services Business Process Execution Language (WS-BPEL)
- Web Services Choreography Description Language (WS-CDL)

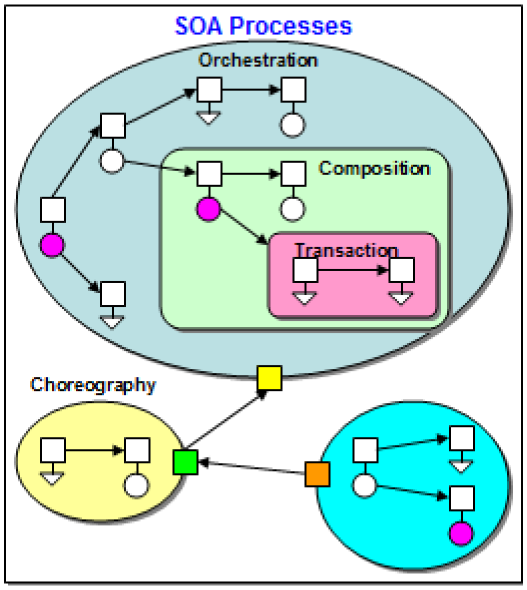


Figure 4.8. SOA Processes View

Formally, we define the orchestration process in the following form:

$SO = \{so_i \mid so_i$ is a service orchestration process$\}$

where

$$so_i = \langle S_{orc}, I_{orc}, M_{orc}, CS_{orc}, C_{orc}, H_{orc}, A_{orc}, Q_{orc} \rangle, \qquad (4.16)$$

in which

$S_{orc} \subseteq S$ is a set of services executed in a certain order defined by BPEL [155];

$M_{orc} = \{ST_{orc}, D_{orc}\}$ in which ST_{orc} is a set of states and D_{orc} is a set of variables as well as data;

CS_{orc} is a set of correlations of messages;

C_{orc} is a set of orchestration process coordinators or controllers;

H_{orc} is a set of handlers;

$A_{orc} = \langle a_{obs}, \tau_{unobs}, \phi \rangle$ in which a_{obs} is a set of observable actions, τ_{unobs} is a set of silent actions performed by the process, and ϕ indicates no action for an idle process; A_{orc} defines the interaction model of an orchestration;

$Q_{orc} \subset Q$ is the set of orchestration process quality attributes. Table 4.5 describes a mapping from model (4.16) to the components in WS-BPEL [155].

From Table 4.5, one can see that the proposed model is an abstraction of WS-BPEL and it is in a concise form which captures its core concepts and parts, such as process participants, interfaces and activities, and extends it with quality attributes.

Visually, the orchestration can be depicted both structurally and behaviourally by UML activity diagrams. Figure 4.9 is an example of sequence service orchestration process modelled by UML activity diagram.

Table 4.5. Mapping to WS-BPEL

Abstract model		**Component in WS-BPEL**
S_{orc}		<partnerLinks> – roles of process participants
I_{orc}		<portType> – the operation interfaces of participants
M_{orc}		<variables> – the data and state used within process
CS_{orc}		<correlationSets> – properties that enable conversations, such as the state of initialization; message invocation patterns: request|response|request-response
H_{orc}		<compensationHandler>, <eventHandler>, <terminationHandler>, <faultHandler>
A_{orc}	a_{obs}	<receive>, <pick>, <onEvent>, <onMessage>, <onAlarm>, <reply>
	τ_{nobs}	<assign>,<compensate>,<flow>,<invoke>, <scope>,<sequence>,<switch>,<while>,<wait>
	ϕ	<empty>
Q_{orc}		Such as <compensationHandler> for transaction; <faultHandler> for reliability

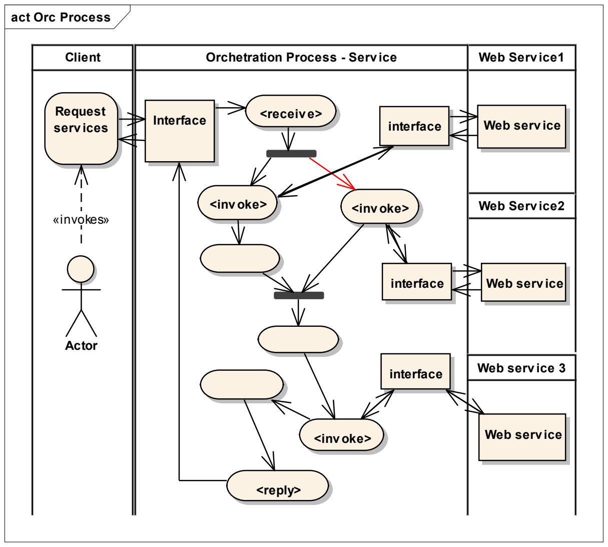


Figure 4.9. Service Orchestration Process

The services choreography can be formally defined as

$$SC = \langle CHO, INF_{cho}, P_{cho}, I_{cho}, L_{cho}, C_{cho} \rangle, \tag{4.17}$$

in which

$CHO = \{ sc_i | sc_i$ is service choreography$\}$

CHO is a set of choreographies for participating collaborations,

where

$$sc_i = \langle CHO_{sub}, M_{cho}, A_{cho}, CR_{cho}, Q_{cho} \rangle, \tag{4.18}$$

In (4.17),

INF_{cho} is a set of declarations of data types for messages;

$P_{cho} \subseteq SO \cup S$ is a set of services and service processes;

I_{cho} is a set of interfaces of participants, which defines the observable behaviours of each participant;

L_{cho} is a set of links among publicly observable participant behaviour – the constraint between two interfaces;

$M_{cho} = \{ST_{cho}, D_{cho}, CH_{cho}\}$ in which ST_{cho} is a set of states; D_{cho} is a set of variables and data; CH_{cho} is a set of information of communication channels, such as URI;

C_{cho} defines a point of collaboration between participants by specifying where and how information is exchanged;

Table 4.6. Mapping to WS-CDL

Abstract model	**Component in WS-CDL**	
SC	<package> – a groups of abstract types	
sc_i	<choreography> – contains other choreographies, variables, activities, rules and exception handler and finalizer	
INF_{cho}	<informationType> – declaration of data types	
P_{cho}	<participateType> – abstract of a participate service, specifically, orchestration process	
I_{cho}	<roleType> – a specification of operation interfaces of participates	
L_{cho}	<relationshipType> – A relationshipType identifies the roleTypes and behaviors, where mutual commitments MUST be made for collaborations to be successful.	
M_{cho}	<variables> – the data, state, channel information used within choreography	
C_{cho}	<channelType> – A *channelType* realizes a point of collaboration between participantTypes by specifying where and how information is exchanged.	
A_{cho}	a_{obs}	<sequence>, <parallel>, <choice>, <interaction>, <perform>, <workUnit>
	$\tau_{:nobs}$	<assign>,<silentAction>
	ϕ	<noAction>
CR_{cho}	collaboration rules and constraints	
Q_{cho}	Such as <exception>, <finalizeBlock> for reliability	

$A_{cho} = \langle a_{obs}, \tau_{unobs}, \phi \rangle$ in which a_{obs} is a set of observable actions performed by the participants, and $\tau_{..nobs}$ is a set of silent actions which are unobservable, but impact globally observable behaviours, and ϕ indicates no action for an idle participant;

CR_{cho} defines choreography collaboration rules and constraints, such as order rules = {sequence, parallel, choice}; exception handling rule; and finalizing rule [230];

$Q_{cho} \subset Q$ is the set of service choreography quality attributes. Table 4.6 shows a mapping from the abstract model to the main components in WS-CDL [230].

Table 4.6 shows that the proposed model is an abstraction of WS-CDL and it is a succinct form in concept and structure which catches its core concepts and parts, such as choreography, process participants, and activities. Moreover, our model extends WS-CDL with quality attributes.

The UML diagram of the service choreography structural model (4.16) is shown in Figure 4.10.

The SC behaviours can be modelled in terms of UML behavioural diagrams, such as sequence diagram and activity diagram. Figure 4.11 shows a choreography example with four participants.

P_{cho} = {Buyer, Seller, PaymentService, ShipService}

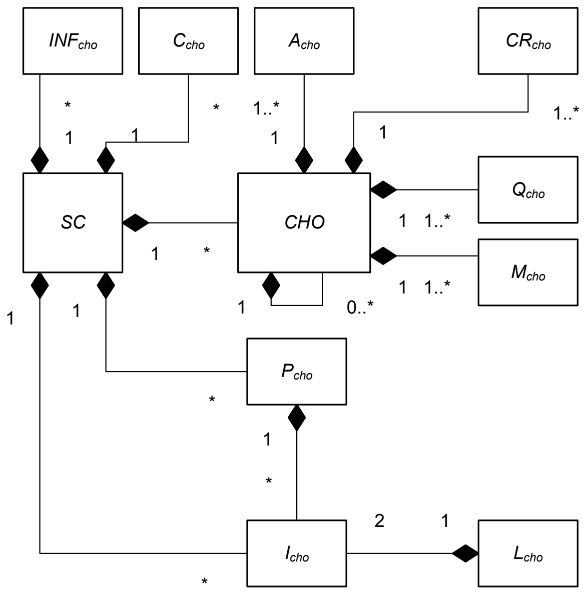


Figure 4.10. Structural Model of Service Choreography

Although one can map the orchestration model to WS-BPEL in Table 4.5 and map the choreography model to WS-CDL in Table 4.6, the proposed models are independent of concrete languages such as WS-BEPL. They can be mapped to other process description languages.

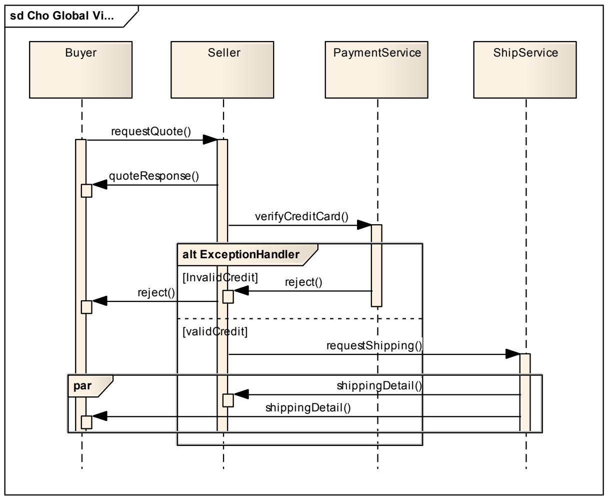


Figure 4.11. Service Choreography Example

4.5.7 SOA Quality Attributes

Quality is an important requirement for software architectural design as well as ESOA system design. Therefore, the SOA quality attributes, *SQ,* are the rationale for the various choices and alternatives in realizing ESOA in enterprise systems. The QoS and SLA are based on the quality attributes. In model (4.2), *SQ* can be defined as a set of SOA quality attributes based on [158]:

$$SQ = \{IN, RE, AV, US, SE, PE, SC, EX, AD, TE, AU, OD, MO\}\ . \qquad (4.19)$$

For an EWS-* style ESOA, the SOA quality attributes are specified and implemented by different service specifications [235]. Table 4.7 lists part of the specifications which are related to SOA quality attributes in EWS-* style systems.

Table 4.7 lists all attribute names and status. The “Status” column refers to the level of maturity of SOA in that area. The green color indicates that there are known solutions for the

SOA based on relatively mature standards and technologies. The yellow color indicates that some solutions exist but need further research to prove their usefulness in handling the requirements for the quality attribute. The red color indicates that the standards and technologies are immature and further significant effort is required to fully support the quality attribute within an SOA. Detailed descriptions can be found in [158].

Table 4.7. EWS-* SOA Quality Attributes

Quality Attribute (abbreviation)	**Web Service Specifications**	**Status**
Interoperability (*IN*)	WS-I Profiles	**Green**
Reliability (*RE*)	WS-Reliability WS-ReliableMessaging	**Yellow**
Availability (*AV*)	No direct specification	**Yellow**
Usability (*US*)	WSDL	**Yellow**
Security (*SE*)	WS-Security WS-Trust	**Red**
Performance (*PE*)	WS-Transaction SOAP Message Transmission Optimization Mechanism XML-binary Optimized Packaging	**Red**
Scalability (*SC*)	Web Service Management Foundation	**Yellow**
Extensibility (*EX*)	WSDL 2.0	**Green**
Adaptability (*AD*)	No direct specification	**Yellow**
Testability (*TE*)	No direct specification	**Red**
Auditability (*AU*)	No direct specification	**Red**
Operability and Deployability (*OD*)	WSDL, WS-BPEL, WS-CDL	**Yellow**
Modifiability (*MO*)	WSDM-MOWS, WS-ResourceMetadataDescriptor	**Green**

4.5.8 Relationship of Parts of EWS-* Style

Sections 4.5.1 - 4.5.7 have specified common structure, behavior, and constraints of all EWS-* style parts in the generic model (4.2) and discussed some of their relationships. Sections 4.5.1 and 4.5.2 discussed the relationship between service consumer and services. The consumer, typically a web service client application, requests services and services serve its request. One can also connect service-level quality attributes to SOA quality attributes (*SQ*). Sections 4.5.5 and 4.5.6 discussed the relationships of SOA Management (*SM*), and SOA processes (*SP*) and SOA quality attributes (*SQ*). Figure 4.12 shows the overall relationship for EWS-*. All EWS-* parts and their constraints are formally and informally described from Section 4.5.1 - 4.5.7. Each part in Figure 4.12 is described by typical elements in enterprise or some of graphics, such as web services. For example,

$C = \{\text{Portal, Thin Client, Fat Client, Wireless Client}\}$.

This section discussed the relationship of all parts described in Figure 4.12. The services (*S*) are registered in service registry of SOA Infrastructure (*SI*) and they are represented by using SOA metadata and data (*D*). Consumers (*C*) request services through the service discovery and communication channel, such as Enterprise Message Bus (ESB) provided by *SI*. The orchestration and/or choreography of Service Process (*SP*) consists of a set of services in *S*, which are managed and controlled by SOA Management (*SM*). Moreover, the *SM* also manages *SI*, *S,* and *C*. *SM* security and policy management and system management rely on SOA Quality Attributes (*SQ*). The monitors in *SM* are monitoring *C*, *SI*, *S,* and *SP*. The quality attributes and constraints based on them in *SQ* are applied to all the other parts. The *D* is used by all the other parts.

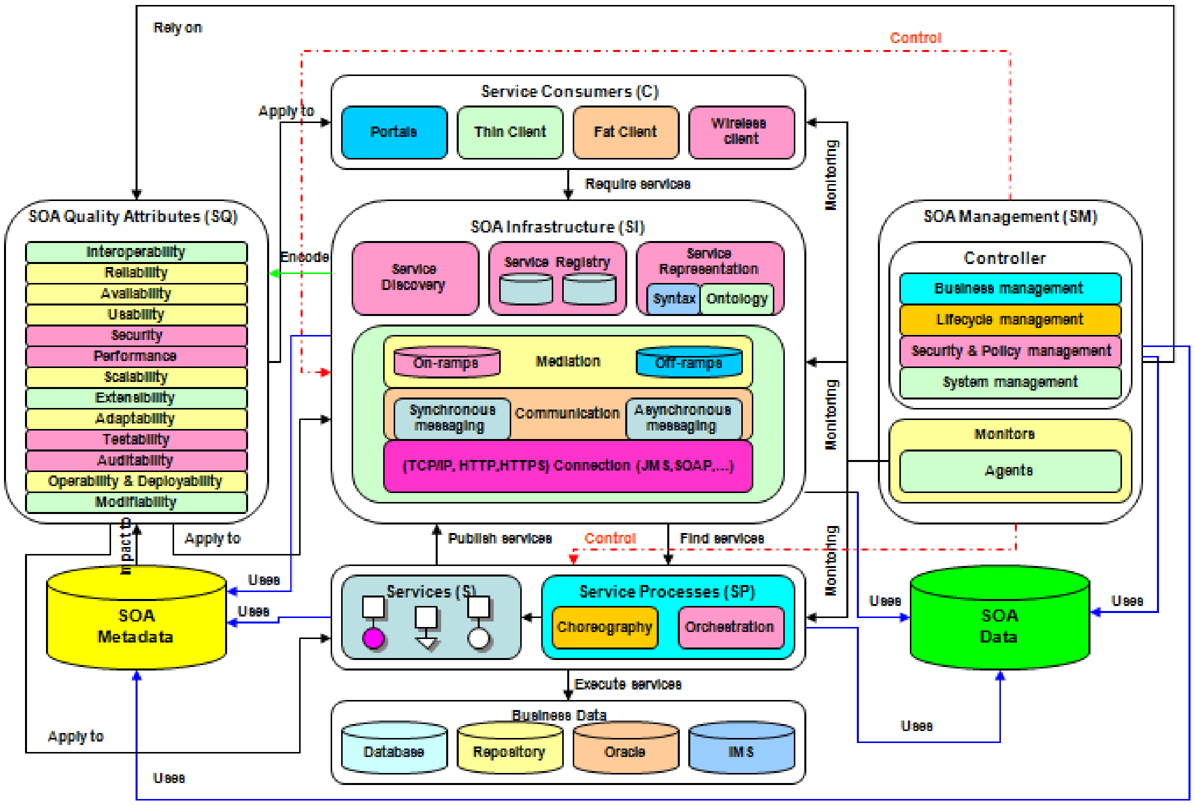


Figure 4.12. ESOA Model for EWS-* Style

4.6 Specifying EWOA

With successful application of Web 2.0 [174] by a lot of new web applications and websites, such as Google AdSense, Wikipedia, blogging, and the emergence of many new web technologies, such as RESTful web services, AJAX, RSS, JSON, Rudy, and Mashup, the Enterprise Web-Oriented Architecture (EWOA) is gaining great attention from both industry and research community. The traditional SOA [66] is an overall umbrella concept and style for how to create the web services with WS-* style, SOAP protocol and WSDL language. The ESOA is a specific style of SOA for enterprise systems [201][204]. However, the web, HTTP protocol, and web browsers do not directly support SOAP and WSDL specifications, and the design and implementation of traditional SOA and ESOA requires complex tools and frameworks because of its complexity. The EWOA is really a push back on the complexity of the traditional EWS-* style ESOA. It is an alternative style for web-centric web services. Figure 4.13 shows what the SOA core with reach - WOA [92] looks like. The traditional ESOA is service-centric instead of web-centric and, thus, can be applied to web-centric and desktop applications. However, the traditional ESOA style does not take advantages of web simplicity for web-centric web services. That is why it is not widely adopted for web-centric applications.

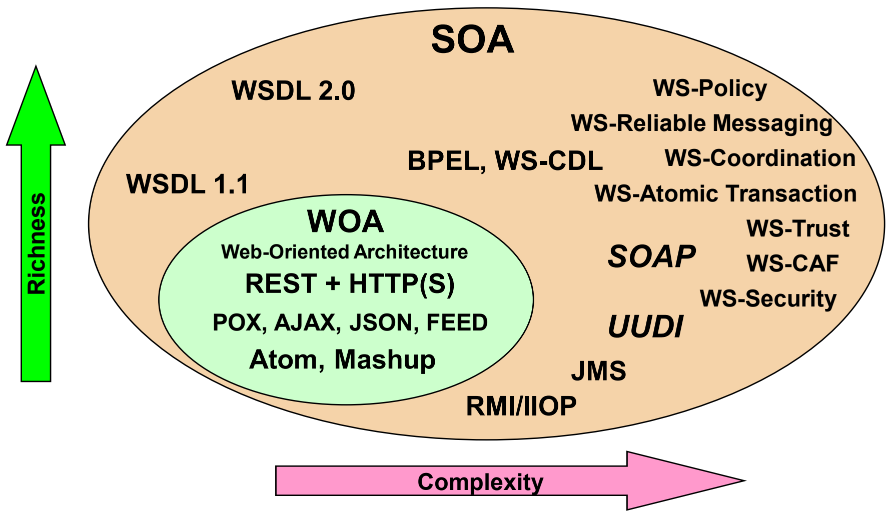


Figure 4.13. SOA Core with Reach – WOA

In this section, we call the WOA for a web-oriented enterprise as EWOA. In Section 4.4, the EWOA is defined as a sub-style of ESOA and a new way to build service-oriented applications on the web has not been well-defined. To introduce it, we use the definition from Cartner's Nick Gall [74]:

> "WOA is an architectural style that is a sub-style of SOA based on the architecture of the WWW with the following additional constraints: globally linked, decentralized, and uniform intermediary processing of application state via self-describing messages."

Nick Gall also gives an interesting mathematical formula for defining WOA as

$$\text{WOA} = \text{SOA} + \text{WWW} + \text{REST} \qquad (4.20)$$

The mathematical formula can be depicted by the WOA triangle shown in Figure 4.14.

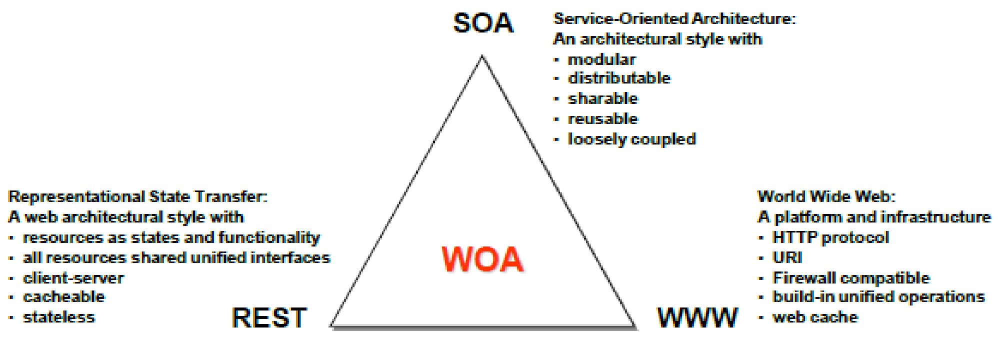

SOA
Service-Oriented Architecture:
An architectural style with
· modular
· distributable
· sharable
· reusable
· loosely coupled
Representational State Transfer:
A web architectural style with
· resources as states and functionality
· all resources shared unified interfaces
· client-server
· cacheable
· stateless
WOA
REST
WWW
World Wide Web:
A platform and infrastructure
· HTTP protocol
· URI
· Firewall compatible
· build-in unified operations
· web cache

Figure 4.14. Triangle of Web-Oriented Architecture

In the WOA triangle, the SOA is the parent architectural style of WOA which is built on many SOA principles, such as statelessness and loosely coupled-ness. The WWW and REST are the base of WOA. The WWW is the platform and infrastructure of WOA. It is a mature global network based on HTTP protocol. The REST (Representational State Transfer) [70] is the foundation of the WOA architectural style. It is a simple web architectural style which is developed as "an abstract model of the Web architecture to guide our redesign and definition of the Hypertext Transfer Protocol and Uniform Resource Identifiers" [70][71]. The model can be formally defined as the following 4-tuples:

$$\text{REST} = < \text{Elements, Principles, Constraints, Quality} > \qquad (4.21)$$

where

Elements = {REST Data, REST Connectors, REST Components}

Principles = {Application states and functionality as resources,

Representation of a resource, Stateless, Layered, Cacheable} (4.22)

Constraints = {Web Platform, HTTP Protocol, URI Addressing, Client-Server, Uniform HTTP Interfaces} (4.23)

Quality = {Performance, Scalability, Simplicity, …} (4.24)

We have defined the Enterprise Service-Oriented Architecture (ESOA) as the set of architectural elements, environments, principles and processes in [201][204]. The EWOA is a sub-style of ESOA. Thus, EWOA is also defined as the sets of web-based architectural elements, environments, principles and processes based on [70] and [71]:

$$\text{EWOA}=\langle S_R, C_R, D_R, S_RI, S_RM, S_RP, S_RQ\rangle, \tag{4.25}$$

in which

$$S_R = \{s_R \mid s_R \text{ is a RESTful web service}\}, \tag{4.26}$$

$$C_R = \{c_R \mid c_R \text{ is a web client}\}, \tag{4.27}$$

$$D_R = \{d_R \mid d_R \text{ is a WOA data element}\}, \tag{4.28}$$

$$S_RI = \{w_R \mid w_R \text{ is a WOA platform}\}, \tag{4.29}$$

$$S_RM = \{m_R \mid m_R \text{ is a WOA management}\}, \tag{4.30}$$

$$S_RP = \{p_R \mid p_R \text{ is a WOA process}\}, \tag{4.31}$$

$$S_RQ = \{q_R \mid q_R \text{ is a WOA quality attribute}\}. \tag{4.32}$$

The formula defines EWOA as a substyle of ESOA. It is based on the generic service-oriented enterprise architectural formula (4.2). We specify the EWOA from Section 4.6.1 - 4.6.7 formally and informally. We discuss the high-assurance EWOA in Section 4.6.8.

4.6.1 RESTful Web Services

The RESTful web services (RWS) is the key element of EWOA. Like a generic service model defined in [204], formally, we can define a RWS s_R as the following 5-tuple:

$$s_R = \langle I_R, M_R, R_R, l_R, Q_R \rangle , \tag{4.33}$$

where

$$I_R = \{i_R | \, i_R \text{ is an HTTP interface}\}, \tag{4.34}$$

$$M_R = \{s_R | \, s_R \text{ is an RWS state}\}, \tag{4.35}$$

$$R_R = \{r_R | \, r_R \text{ is a web resource}\}, \tag{4.36}$$

$$l_R = \{u_R | \, u_R \text{ is a URI}\}, \tag{4.37}$$

$$Q_R = \{q_R | \, q_R \text{ is a service quality attributes}\}, \tag{4.38}$$

Formula (4.34) indicates that the RESTful web services have uniform interfaces which are HTTP GET, POST, PUT, DELETE, HEAD, OPTIONS, TRACE, and CONNECT based on HTTP 1.1. For most enterprise web applications, the first four interfaces cover almost every operation as shown in Table 4.8.

Table 4.8. Uniform Service Interfaces

HTTP Interface	Semantics in RESTful Web Services
GET	Retrieve information from resource
POST	Add new information
	Show its relation to old information
PUT	Update information
DELETE	Discard information

Formula (4.35) shows that an RWS has a set of states maintained as part of the content transferred from client to server and then back to client. The set of states includes Application state, which is the information for the server to understand how to process the request. The authorization and authentication information are examples of application states. It includes Resource state, which is the representation of the values of the properties of a resource.

Formula (4.36) indicates an RWS serves a set of resources which are application states and functionalities of the RWS. Formula (4.37) tells us that an RWS can be described by a set of URIs each of which is a single string including the service address and the specification of the resource. For example, a service for browsing all books URI looks like

http://www.amazon.com/books

Formula (4.38) is a set of RWS service quality attributes which include performance, scalability, simplicity, etc. The detailed analysis of these attributes is presented in Section 4.6.7.

Algebraic Characteristics of Set (4.24): For any resource $r \in R_R$, there exists one or many URI in l_R for the resource. If resources r_1 and $r_2 \in R_R$, then only one statement will be true: $r_1 \neq r_2$ or $r_1 \equiv r_2$. It shows that the same resources or the same URIs have the same behavior or result to the client. Therefore, a non-POST RWS is idempotent.

We propose an abstract tuple model (4.25) of RESTful web services. Figure 4.15 presents the relationship between sets in (4.25) and structural and behavioral models of RWS. The relationship between set (4.35) through set (4.37) can be summarized as follows:

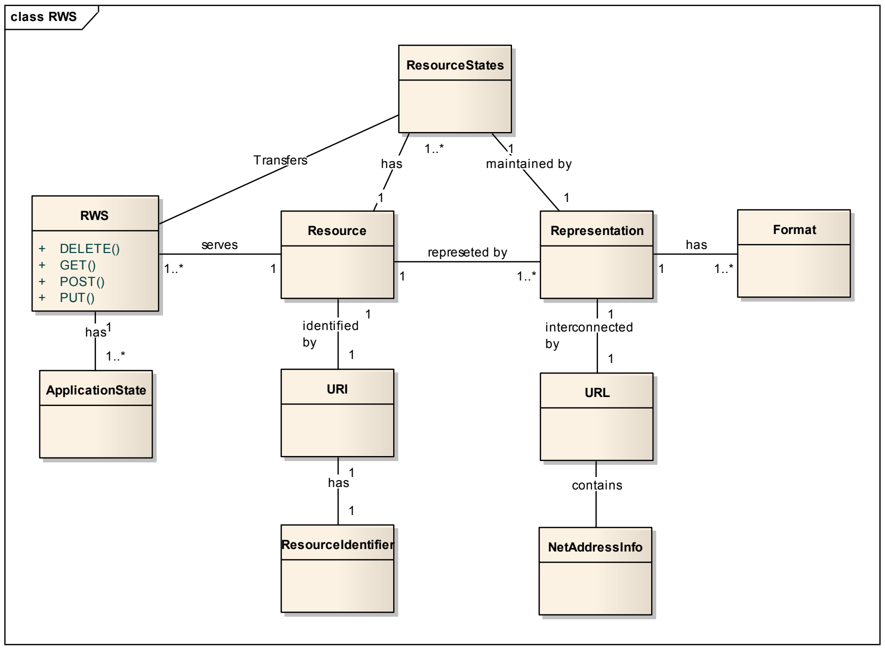


Figure 4.15. Relation Model of RWS

- An RWS, with application states, serves resources by processing requests and transfers resource states from one to another in term of response.
- A resource, which is a conceptual entity, can be represented by many representations which are concrete manifestation of the resource.
- A resource has one unique URI and many resource states. Each state is maintained by the resource representation.
- A URI has the resource identifier.
- A resource representation can be located by a URL with network address and other information which includes the protocol (http or https), hostname, path and extra information for describing how to get the representation of a resource.
- A resource representation can be represented by multiple formats, such as XML, HTML, and JSON.

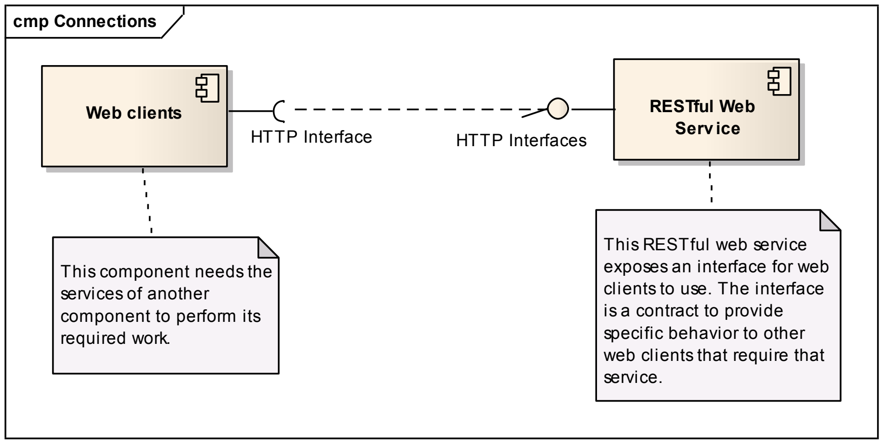

cmp Connections
Web clients
HTTP Interface
HTTP Interfaces
RESTful Web Service
This component needs the services of another component to perform its required work.
This RESTful web service exposes an interface for web clients to use. The interface is a contract to provide specific behavior to other web clients that require that service.

Figure 4.16. Connection Model of RWS

Figure 4.16 shows the RWS connectional models. We leave the discussion of RWS behavior model in the next section.

4.6.2 RESTful Web Service Consumers

According to the connection model of RWS in Figure 4.16, any web client can be the consumer of RESTful web services. For each $c_R \in C_R$, it has the following behaviors:

- Connect to web services by HTTP protocol;
- Send RESTful requests through RESTful interfaces;
- Consume RESTful web services in WWW browsers or any web application.

There are two interaction models, which describe how web clients consume RESTful web services:

- Synchronous interaction model

The Java JDK HttpURLConnection [90], Apache's HttpClient [17], and Microsoft's WebHttpBinding of WCF [69] all provide the client model for accessing RESTful web services synchronously. The model is based on HTTP request and reply model. The sequence diagram in Figure 4.17 depicts the model.

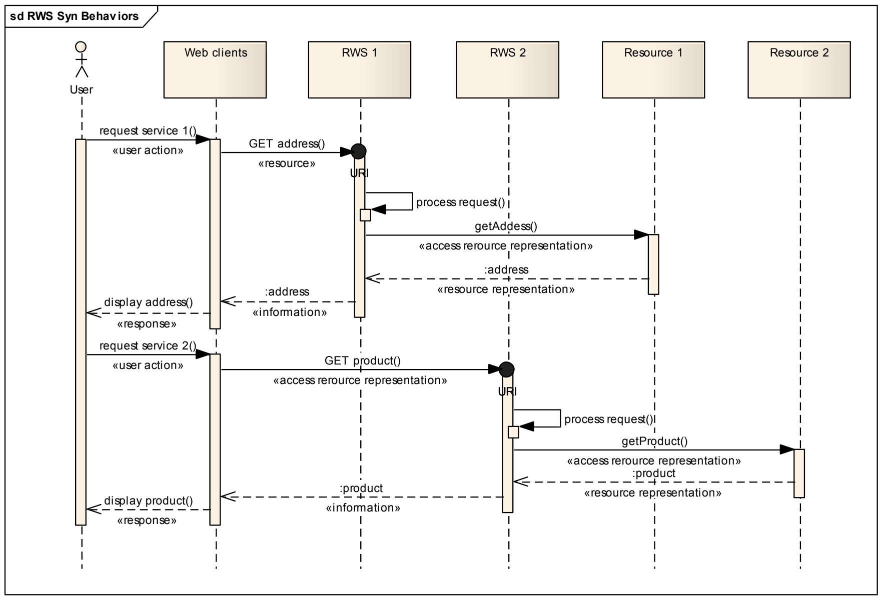


Figure 4.17. Synchronous Interaction Model

The UML 2.1 sequence diagram depicts two RESTful web services RWS 1 and RWS 2 which serve two user requests: GET address and GET product for a shopping page on the web. To best describe the behaviors of RESTful web services, we create a RESTful profile with the following stereotypes:

<<user action>>

<<resource>>

<<access resource representation>>

<<resource representation>>

<<information>>

<<response>>

which are helpful in describing the interaction behavior between service consumers and RWS. They are also used in the UML sequence diagram for describing the following asynchronous interaction model:

- Asynchronous interaction model

The EWOA uses HTTP which is a synchronous request/response protocol. The question is whether the EWOA can support asynchronous interactions for long-running processes. In fact, there exist some standard asynchronous interaction patterns supported by HTTP, which are independent of the RESTful web services approach. The patterns are listed in Table 4.9.

Table 4.9. Standard Asynchronous Interaction Patterns

Asynchronous Patterns	Description
Reliable one-way messaging (Fire-and-forget)	Service consumer does not wait for response
Polling	Service consumer periodically polls the request status
Callback	Service provider calls consumer back when service is done

In EWOA, the web clients can interact with RWS asynchronously by using AJAX which is a set of technologies including the asynchronous JavaScript and XML [174]. The UML sequence diagram in Figure 4.18 shows such a model.

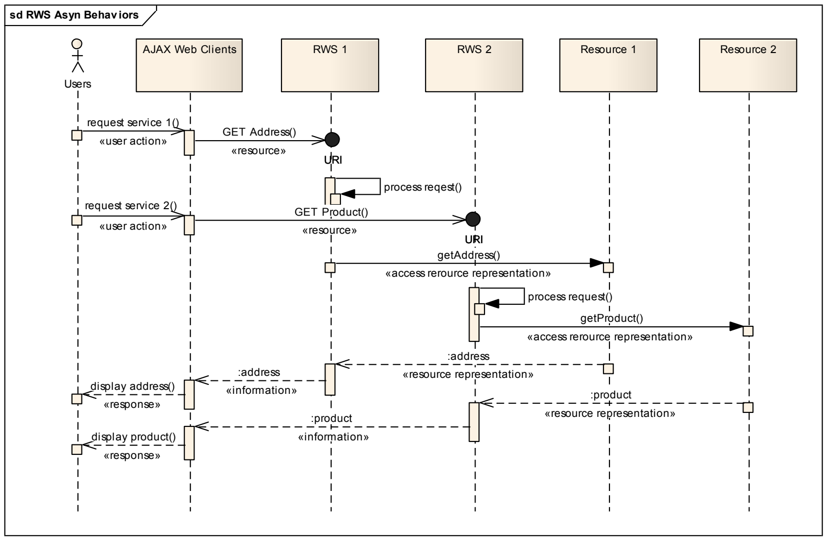


Figure 4.18. Asynchronous Interaction Model

The sequence diagram shows that the user can submit two service requests to two RWS almost in parallel to update web page blocks and without going to the web server and refreshing the page for each request.

4.6.3 WOA Data Elements

As a RESTful architectural style, the D_R in the model (4.25) plays an important role for understanding, specifying, and designing WOA systems. The D_R is a finite set which consists of certain abstract data types supported by the style. They can be informally defined as shown in Table 4.10.

Table 4.10. WOA Data Elements

Data Elements	Specification
Resource	The intended conceptual target of a hypertext reference [9], such as an online address book and a shop invoice
Resource metadata	The data for specifying a resource, such as a source link
Resource identifier	URI and URL
Representation	The current or intended state of a resource, such as HTTP document, XML document, and JPEG image
Representation metadata	The data for describing the representation, such as Media type, last-modified time
Service specification	WSDL 2.0 RESTful web service specification
WOA metadata	The data for describing other metadata, such as message integrity and service quality contracts
WOA Management data	Security policy data
WOA process data	Workflow description
Web configuration data	Configuration of Web servers, DNS, Server Proxy, Gateway, Cache
Web container data	Configuration of application server web container, such as weblogic web container

In Table 4.10, the first five rows, such as Resource and Representation, are REST data [70] which are the base of WOA data elements.

4.6.4 WOA Infrastructure

Unlike other ESOA substyles, EWOA is built on existing web infrastructure in the enterprise. The $S_R I$ in (4.25) can be defined as a set of servers and services:

$S_R I$ = {Web servers, Proxy servers, Gateway, DNS, Server connectors, Cache servers, Web containers of application servers}. (4.39)

For small and some medium enterprises, the WOA infrastructure is a subset of $S_R I$. For example, they may not have application servers, even Proxy servers. Formula (4.39) describes the major components in a generic EWOA infrastructure. The role and functionality of each infrastructural component are defined in Table 4.11.

Table 4.11. Role and Functionality of Infrastructural Components

Infrastructural Components	Example	Role and functionality
Web servers	Apache HTTP server, and IIS	HTTP communication, service request and response processing, HTTP security, Cookie, session management
Proxy servers	SUN' SQUID	HTTP server routing, RESTful web service routing
DNS	***Round*** Robin DNS	URI addressing
Gateway	CGI	RESTful web service provider
Web Containers	java web container	RESTful web service provider
Server connectors	Libwww, JDK, NSAPI, .NET, DNS lookup, Tunnel (such as SOCKS, SSL)	Make connection between client and server
Cache service or servers	Browser cache, JCache, Akamai Cache Network	Store short-life data for improving performance

4.6.5 WOA Management

The EWOA is the WOA for enterprise, so it also includes WOA management $S_R M$ which is a set of web application system management tools and services for managing RESTful services. The $S_R M$ includes

- RESTful web services registry;
- Firewalls for network security management, such as Perimeter firewall, NAT firewall, XML firewall;
- Filters for request and response management, such as Java HTTP filter;
- Security services for application security management, such as authentication, authorization, REST parameter analysis and XML threat analysis;

- Logging services for error and exception management;
- Agents and Monitors for performance management.

We will discuss the importance of WOA management for high-assurance RESTful web service computing in Section 4.6.8.

4.6.6 WOA Processes

Traditional web service architectures are designed to accommodate simple point-to-point interactions – there is no concept of a logical flow or series of steps from one service to another. In an enterprise, the business often requires software systems to have the capacity to process complex business processes, such as workflow, transaction, online order, and shipping. Supporting service composition (orchestration and choreography) is fundamental to the web services vision. Therefore, the service processes are one of the core elements in ESOA [201][204]. As mentioned in Section 4.5.6, there are two specifications, BPEL and WS-CDL, for handling the different approaches of orchestration and choreography of SOAP-based web services in traditional EWS-* style ESOA for various complex business process management. Although there is no corresponding standard for EWOA processes, RESTful web services composition, such as client-side or server-side Mashup, have been practiced on the Web. iGoogle is a good example. The Web is the most complex global enterprise on the business platform. To meet the increasing number of requests for handling complex web business processing and services interactions, many software industry vendors and researchers are working on specification and tools for WOA processes of both RESTful orchestration and choreography. The Bite is a minimalist choreography language for Web [54]. The Bite runtime architecture is implemented by IBM Project Zero [97]. An approach to RESTful process choreography based on the Asynchronous Services Access Protocol (ASAP) is proposed in [146]. There are several approaches to RESTful process orchestration [17][176]. A common idea is to extend BPEL for RESTful web services orchestration. Figure 4.19 depicts how to extend BPEL for two RESTful web services s_R^1 and s_R^2 orchestration.

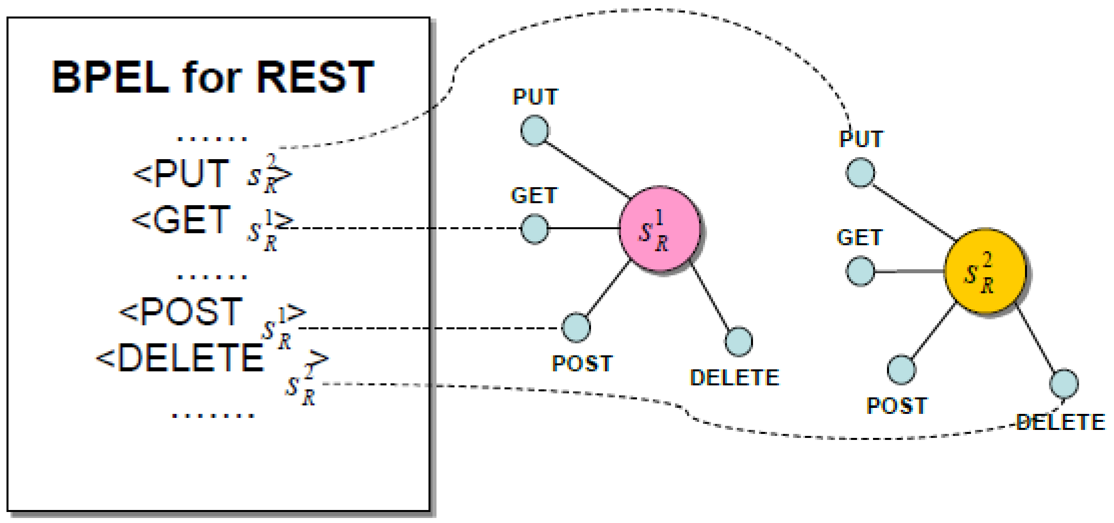


Figure 4.19. RESTful Web Services Orchestration by Extended BPEL

4.6.7 WOA Quality Attributes

The quality attribute requirements drive high assurance software architecture design [158]. They also drive the ESOA and EWOA system design for high assurance. In this section, we define a set of quality attributes as architectural properties of EWOA style. The REST and the Web are two bases of WOA. The quality attributes of both WEB and REST are discussed in [70]. We list the major parts in Table 4.12.

Table 4.12 describes the basic quality attributes of the WOA style. For the EWOA which is an enterprise-level WOA style, we have to address additional non-functional requirements for some of the quality attributes, such as security, reliability, manageability, governance. We define high-assurance EWOA style which can address further enterprise non-functional requirements.

Table 4.12. Quality Attributes of WEB and REST Style

Quality Attributes	Description for WEB and REST
Performance	Network performance which is one of infrastructure performance which can be improved by interaction style
Efficiency	REST is cacheable. Using cache can improve application performance and network efficiency
Scalability	WEB is internet-scale
	Using proxy style can increase web scalability
Simplicity	REST is very simple style by client-server for separating concerns
Security	HTPS, SSL, firewalls provide basic WEB infrastructure security. REST does not address application security.
	Firewall visibility increases security, but visibility may reduce payload level security.
Evolvability	WEB is easy to evolve. REST style can improve web architecture evolvability.
Extensibility	REST supports the gradual and fragmented deployment of changes within an already deployed architecture
Reusability	The components defined by REST are reusable
	REST style use uniform HTTP interfaces
	Sharable proxy and cache style all increase reusability
Reliability	REST style can help reliability by avoiding single failure point, enabling redundancy, using monitoring, or reducing scope of failure to a recoverable action.
Visibility	"Within REST, intermediary components can actively transform the content of messages because the messages are self-descriptive and their semantics are visible to intermediaries."
Modifiability	REST style also improves system modifiability through supporting evolvability, customizability, configurability and reusability.
Customizability	It is induced by remote evaluation and code-on-demand style
Configurability	WEB Servers and other mediators, such as proxy are configurable.

4.6.8 High-Assurance EWOA

To achieve high-assurance SOA in the enterprises, specifically for defense, financial industry, and mission critical business systems, the traditional ESOA style addresses the enterprise architectural non-functional requirements or quality attributes through the WS-* standards [235] and governance framework. They are presented in our previous work as a set of SOA managements [201][204] which can be governed by QoS rules and policies. Therefore, the system based on traditional ESOA-style is very complex in general. The WOA is a lightweight approach to SOA at Web, so it greatly reduces the complexities of SOA with its two fundamental: REST style and mature Web infrastructure. Because of its simplicity, EWOA does not need WS-* like complicated governance and management. However, to meet enterprise requirements for high-assurance service computing, such as web transaction, e-Business of inter-organizations and inter-business partners, dynamic web information system integration, EWOA needs RESTful governance. The SOA governance includes design time governance and runtime governance. In this chapter, we focus on specifying the EWOA-style runtime governance which is what we have defined as WOA management in Section 0. In our specification, the RESTful lightweight governance may include:

- RESTful services registry/repository;
- RESTful security management;
- RESTful application controller, such as a java servlet;

We propose the high assurance RESTful information system architecture as shown in Figure 4.20 based on the EWOA style we have specified.

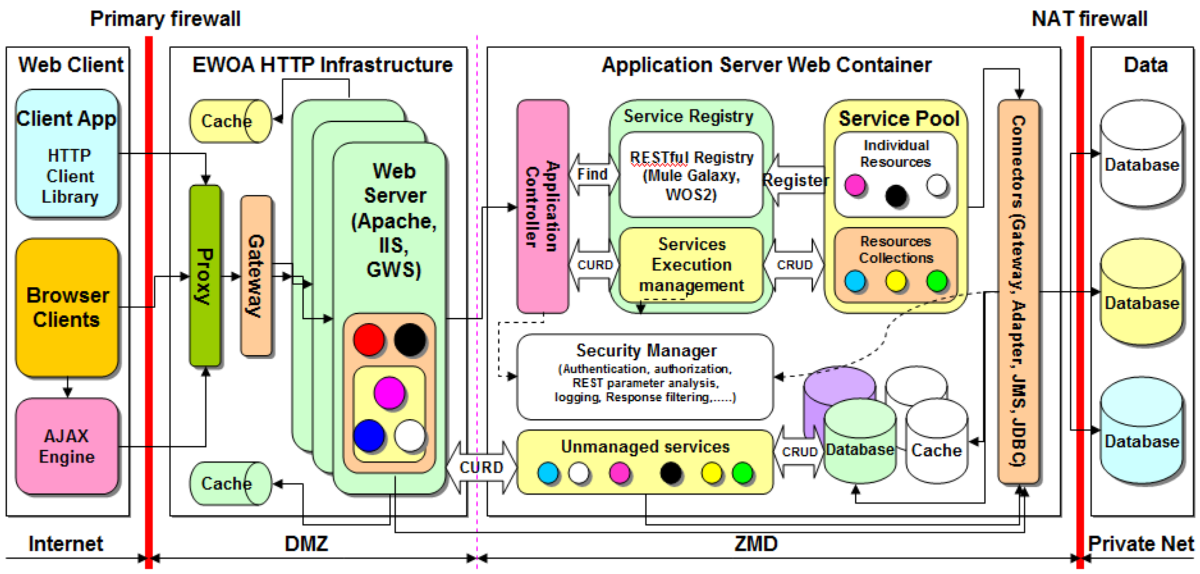


Figure 4.20. High-Assurance RESTful Information System Architecture

The RESTful architecture consists of the following parts:

- A set of web clients which includes any client application by using HTTP client library and any web site with or without AJAX.
- An EWOA HTTP infrastructure which includes a set of web servers and services, such as web servers - Apache, IIS and GWS, and services - proxy, gateway, web cache. The EWOA infrastructure also includes a set of data source connectors, such as Adapters, JMS, and JDBC.
- A set of RESTful services which can be served by two kinds of resources - individual resources by GET, PUT and DELETE interfaces and resource collections by GET and POST interfaces. We define two kinds of RESTful web servers:
  - Managed RWS which is registered by the service registry;
  - Unmanaged RWS which is for getting public data only;
  - The RWS can be deployed in either the web server extension, such as secure cgi-bin, or web containers, such as weblogic and Tomcat.

The EWOA management consists of an Application Controller, a Security Manager, and a Service Registry which includes a repository for storing the description of RWS and policy as well as configuration data, and server and application monitors. The controller can also act as an RWS orchestration engine.

Due to the simplicity of the RWS and the architectural properties of REST style, we point out Table 4.12, EWOA style system is of higher performance and simplicity compared to traditional WS-* SOAP style ESOA system. However, the security of RESTful applications for enterprise should be taken into consideration to achieve high-assurance service computing. As we know, the RWS only support four interfaces GET, POST, PUT, and DELETE. Let us define three sets of operations:

A = {a | a is an idempotent and safe operation};

B = {b | b is an idempotent operation either safe or unsafe};

C ={c | c is a non-idempotent and unsafe operation};

O = {o | o is an operation either idempotent or non-idempotent};

Then we have the following security relationship:

$GET \in A$, $PUT, DELETE \in B - A$, $POST \in C$ and

$A \subset B$, $C \subset O - B$.

Figure 4.21 depicts the relationship and exposes the security concerns.

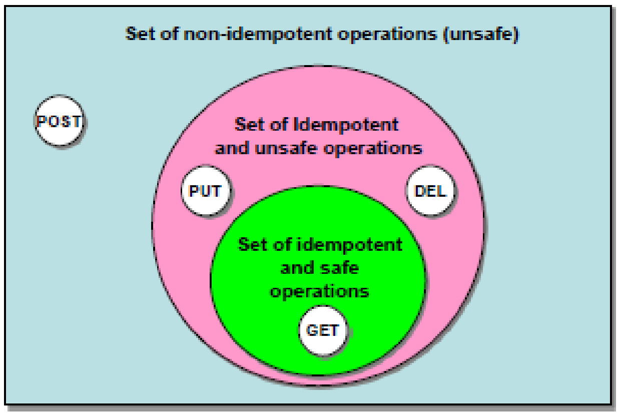

Set of non-idempotent operations (unsafe)
POST
Set of Idempotent and unsafe operations
PUT
DEL
Set of idempotent and safe operations
GET

Figure 4.21. Venn Diagram of RESTful operations

- Except for GET, all the other operations are unsafe. Even GET has some security vulnerabilities, such as QueryString attack and XML/JSON out attack. Unlike SOAP, at the message level, RESTful services are using plain text html for request and POX or JSON for response. Therefore, they do not provide payload-level security for routing RESTful requests to multiple different servers, such as proxy, gateway, web servers, and web containers. Public data which can be accessed by the world;
- Internal confidential data which can be accessed by certain people;

- Business data which can only be accessed by authenticated and authorized users.

Table 4.13 shows a security and QoS comparison between REST message and SOAP message.

- From the example in Public data which can be accessed by the world;
- Internal confidential data which can be accessed by certain people;
- Business data which can only be accessed by authenticated and authorized users.

Table 4.13, the customer's credit card information is in the insecure REST payload. Nevertheless, it can be protected by SOAP envelope at the payload level. In general, the data of any enterprise can be categorized as:

- Public data which can be accessed by the world;
- Internal confidential data which can be accessed by certain people;
- Business data which can only be accessed by authenticated and authorized users.

Table 4.13. Comparison of REST and SOAP Messages

Message	REST POST	SOAP POST
Header	There is no QoS defined in header	Can specify QoS in header
Body	Payload in plain text (HTML or XML), which is visible to cross all traveling servers	Payload inside SOAP Envelope, which is visible only for the end application.
Envelope	There is no Envelope for payload	There is SOAP envelope for payload
Example	**POST/HTTP/1.1** **Host: http://www.amazon.com** **Book: RESTful Web Service** **Credit Card: Visa** **Number: 123456789** **Expire: 11-01-20-12**	**POST/HTTP/1.1** **Host: http://www.amazon.com** **Contenttype: application/soap-xml** **Charset=uft-8** **<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope"> <env:Header>** **<!--Header information here-->** **</env:Header> <env:Body>** **<!--Body or "Payload" here, a Fault if error happened -->** **</env:Body> </env:Envelope>**

In our proposed architecture shown in Figure 4.20, the Security Manager includes authentication which is against identity, and authorization which is against service policy, URI analysis, response filtering, and logging. For the second and third category of data, we always need to use a security manager with SSO (Single Sign-On) and ACL (Access Control List) technologies, where the ACL allows an application to set the data access control for different users. For RWS, we can set the permission to use different operations for different users. For accessing business critical data, such as user account information and transaction data, it is better to use SOAP style web services. However, the RESTful approach has bigger performance and simplicity advantages than WS-* SOAP approach for accessing the public data, specifically by getting them by GET. The unmanaged RWS can serve this kind of data in a very cheap way. In the next section, we discuss the relationship between EWOA and ESOA. Moreover, a hybrid approach is proposed.

4.7 Comparison of ESOA Styles

This section compares the major ESOA styles based on (4.2). The EEDA and EGSA are based on traditional web services in Table 4.14 and Table 4.15 which means the services in EEDA are web services and the services in EGSA are extensions of web services. We show an informal comparison of major quality attributes [158] for all five major ESOA styles in Table 4.14 in which "-" means less, "--" means much less, "---" means dramatically lower, "+" means average, "++" means high and "+++" means dramatically higher. We discuss each of the quality attributes as follows:

Performance: The performance of SOAP web service in enterprise systems is challenging because of its heavyweight nature and XML large payload. In general, the services in EEDA are inefficient [208]. To access web resources, EWOA is more efficient than traditional web services because of its cache feature. Since EGSA's SOA Grid infrastructure enables middle-tier caching layer, it enhances web service performance [39].

Transaction: The ACID (*atomicity, consistency, isolation, durability*) transactions that span multiple applications are important for enterprise systems. EWS-* and other web service based styles are addressed by WS-AtomicTransaction which relies on WS-Coordination specification. ECBS, such as stateless EJB service model, also handles transactions well. EWOA is not good enough to handle ACID transactions, because of its resource-oriented nature.

Interoperability: It is one of the main goals of web services. WS-I profile addresses web service interoperability. However, cross-vendor interoperability of traditional web services (like EWS-*, EEDA, EGSA) is not perfect (only "+"). The interoperability of ECBS is not as good as other styles, since its components are less standardized. EWOA interoperability is based on web principles [70].

Scalability: The tier-based ESOA architecture, such as the EWS-* and ECBS architectures, with traditional middleware, does not scale linearly and does not allow applications to scale on-demand. The EWOA architecture has better scalability than traditional web service SOA architecture because of web scalability nature, cache adoption, and unified interfaces. However, its scalability is also not dynamic and linear. The EEDA also improves the scalability of traditional SOA because of its event-driven asynchronous nature. The EGSA can provide predictable and truly linear scalability because grid computing can help ESOA system to improve the scalability bottlenecks across SOA parts [39].

Security: The security of EWOA is based on HTTP security features and Security Socket Layer (SSL). The ECBS security is not only based on HTTP security and SSL, but also based on component security standards, such as J2EE security standards. Other styles based on web services provide more security capacities through WS-* security specifications [235], such as WS-Security, WS-Trust, and WS-SecureConversation.

Availability: Unlike other styles using redundancy technology for high availability, the EGSA uses SOA Grid to achieve extremely high availability. SOA grid can provide continuous availability and reach 100% active-active server failover. It can also prevent single points of failure and enable automatic service load distribution and a full-range of QoS levels for stateful services and service orchestration. In addition, it can increase throughput and self-healing management as well as SLA enforcement.

Reliability: EWOA has less reliability since the style does not address many enterprise challenges through standards. The styles based on web services are of higher reliability by implementing WS-* reliability specifications, such as WS-ReliableMessaging.

Simplicity: EWOA is the simplest style of the five styles, since it is protocol dependent and its RESTful web services have unified interfaces and its infrastructure is built on existing web infrastructure [202]. Therefore, it is called lightweight SOA. EWS-* is more complex than EWOA and ECBS because it is protocol independent and based on complicated multiple specifications [235]. EEDA is more complicated than EWS-* because of its event processing and its real-time environment. EGSA is the most complex due to its complicated grid environment, stateful web services, and the complexity of managing states in SOA applications.

Flexibility: The ECBS is less flexible than other styles since it is not based on common standards and since it has relatively tight couplings. EEDA and EGSA have higher flexibility, since the event processing is added to EEDA SOA and flexible grid computing infrastructure with predicable scalability and continuous availability.

Business Agility: The ECBA has less business agility than other styles because of its relatively tight coupling.

Resource Manageability: The resource manageability of EGSA is the best since grid computing brings maximum resource utilization, such as virtualization, into ESOA architecture.

Consciousness: All ESOA styles guide building enterprise service-oriented software systems. However, most of the traditional styles systems, such as EWS-*, EWOA, and ECBS systems, are comatose, meaning they are unaware of their surroundings. They cannot independently act on conditions without instruction from a central controller or the aid of human administration. However, EDA of EEDA brings consciousness into the enterprise SOA systems. With the right mix of smart event processing and rules, the EDA enables the ESOA system to consciously react to internal and external conditions that affect the business within a real-time context [208].

Loose coupling: EEDA is the most loosely coupled architecture because of its event-driven nature and asynchronous communication protocol [208].

4.8 Summary

The chapter defines ESOA ontology based on the modeling framework built in Chapter 3. However, description of the ESOA ontology and specification of its sub-styles are based on the previous research work [201][202][203][204]. The ACME-Like language defined in Chapter 3 can be used for modeling ESOA as well. Let us replace Infrastructure(t) with SOA Infrastructure; replace Management(t) with SOA Management; replace Process(t) with SOA Process; replace QA with SOA QA in (3.1). We can define

$$\text{ESOA-EAType} \subseteq \text{SOA Infrastructure} \cup \text{SOA Management} \cup \text{SOA Process} \cup \text{Configuration} \cup \text{Component} \cup \text{Connector} \cup \text{Port} \cup \text{Role} \cup \text{SOA QA}. \qquad (4.39)$$

The Dessertation will not discuss them in detail. Since the Infrastructure, Management and Process in traditional ESOA are basically static (except EGSA – which introduced dynamism to SOA), on-premise, and dedicated, that means for a given period of time, the system topology, management and process are basically static, so they are not constantly varying with time t. The dissertation defines and describes Enterprise Cloud Service Architecture (ECSA) style which is a dynamic EAS in the next chapter.

Table 4.14. Major Quality Attributes Comparison

SOA Quality Attributes (***SQ***)					
Style	EWS-*	EWOA	EEDA	ECBS	EGSA
Performance	-	+	-	+	+
Transaction	++	-	++	++	++
Interoperability	+	+	+	-	+
Scalability	+	++	++	+	+++
Security	+	-	+	+	+
Availability	++	++	++	++	+++
Reliability	++	+	++	++	++
Simplicity	-	++	--	+	---
Flexibility	+	+	++	-	++
Business agility	+	+	+	-	+
Resource manageability	+	+	+	+	+++
Consciousness	-	-	+++	-	+
Loose coupling	++	++	+++	-	++

Table 4.15. Comparison of Parts and their Constraints of ESOA Styles

Style	Services (***S***)	Consumers (***C***)	SOA Data (***D***)	SOA Infrastructure (***SI***)	SOA Management (***SM***)	SOA Process (***SP***)
EWS-*	• SOAP-based web services • Machine-processable description of services, such as WSDL • Machine-processable description of composition of services, such as BPEL • Address challenges by WS-* specifications • Standards - WS-*[235] • Connectivity and interaction o Application protocol – SOAP o Transport protocols – HTTP, TCP, SMTP, JMS,MQ,IIOP o Both RPC-style and messaging communication o Both Request-response and one-way o Both synchronous and asynchronous service invocation • Request/action driven • Different applications expose different interfaces • Contract-first and contract-last • Stateless	• Web service client applications • Web applications	• Metadata of Web services • UDDI keys, data sources • WSDL documents • XML Schema (XSD) • Service description • Endpoint schema • Administrativ e metadata • WS-CDL document • BPEL, XLANG documents • SLA data • XACML documents	• Layered infrastructure [201][] o Connection layer o Communication layer o Mediation layer • SOA enabled middleware [214] • Enterprise Service Bus (ESB) [38]	• Resource management • Service and infrastructure discovery • Network and application monitors • Policy enforcement • SLA management • Exception management • Closed-loop government • Service lifecycle management	• Web service orchestration[169] • Web service chorography[169] • Process coordination scheduler required • Process is triggered by consumer
EWOA	• RESTful web service • Need to model system as resources • Not all challenges are tackled • Web standards[70] • Connectivity and interaction o Application protocol – HTTP o Transport protocol – HTTP/REST-style communication o Both synchronous and asynchronous service invocation • Request or action driven • Different applications expose its resources through the same interfaces: GET,POST,PUT and DELETE [202] • Contract-less • Stateless	• Web applications • Customers or users who interact with services, such as Wiki, directly. [41] defines this kind of SOA as Consumer-Centric SOA (CCSOA) or User-Centric SOA (UCSOA).	• Resources • Resource metadata • Resource representation • Resource identifier • Service description • Configuration data	• Web servers • Web proxy servers • Load balancer • DNS • Gateway • Web container • Server connector • Cache	• RESTful WS registry • Firewall server • Single Sign On security manager • Logging service • System monitor	• RESTful web service orchestration, such as Mashup • Lack of standard • Not support chorography

Table 4.15 continued

EEDA	• Traditional SOA Web services • EDA web services[208] • Standards: WS-Eventing[227] • Connectivity and interaction • Support web service connectivity • Both synchronous and asynchronous service invocation • Event-Driven • Support real-time business • Stateless	• Traditional WS client applications • EDA applications	• Traditional SOA data • EDA data • Event metadata	• Traditional SOA infrastructure • Traditional SOA message infrastructure with pub/sub message queues (ESB) • EDA infrastructure ○ Global Event Listeners ○ Global Event Processors ○ Global Event Producers ○ Complex Events Processing (CEP)	• Traditional SOA management • EDA events system management	• Event-Driven Web service orchestration – support both traditional orchestration and publish/subscribe • Events-Driven Web service chorography • Support both scheduled and unscheduled process coordination • Process is triggered by consumer or producer
ECBS	• Component-based Services • RPC-style, such as RMI communication • Standards: Service Component Architecture standards[153] Java Business Integration[210] • Connectivity and interaction • RPC, RMI, JMS • Request-response • Both synchronous and asynchronousservice invocation • Request or action driven • Stateless	• Web client applications • Desktop applications • Wireless client application	• Traditional component data • Service component metadata • Policy data • Configuratio n data	• Traditional SOA infrastructure • Traditional component infrastructure	• Traditional SOA management • Traditional component-based management • Security management (SSO, ACL) • Performance management	• Component service orchestration • Component service chorography • Process coordination scheduler required • Process is triggered by consumer
EGSA	• Grid Web services-extension of traditional web services • Standards: WS-Resources [157] WSRF [157],OGSA[101], OGSI[160] • Connectivity and interaction • Support web service connectivity • Request-response • Request or action driven • Stateful	• Web service client applications • Grid client applications	• Traditional SOA data • Grid data • Web service resource properties • WSRF Specification • Grid service registry data • Policy data • SLA data	• SOA Grid infrastructure • Grid-enabled middleware • Grid-enabled ESB • Grid service registry • Scheduling service • Job-submit service • Data grid • Enterprise Data Bus • Visualization resources	• Traditional SOA management • Grid management • State management • SLA management • Grid resource management • Self-management	• Grid Web service orchestration • Grid Web service chorography

CHAPTER 5

ENTERPRISE CLOUD SERVICE ARCHITECTURE

This chapter introduces and specifies a new enterprise architectural style – enterprise cloud service architecture (ECSA) based on the framework defined in Chapter 3 and previous research work in [205][207]. ECSA is a hybrid architectural style with ESOA and ECC. In this chapter, the combined style is defined and modeled. Since the previous chapter has specified ESOA, this chapter focuses on modeling ECC. The ECSA quality ontology is defined and analyzed.

5.1 Introducing ECSA

With the globalization of the economic environment, the increasing complexity of business processes makes the enterprise information systems complicated. Enterprise service-oriented architecture (ESOA) is designed to tackle the complexity and build better architectures and solutions for enterprises. Conceptually, the ESOA is an architectural style which defines the concrete ESOA architecture as a set of well-defined services. It can be further abstracted to process layers and composite applications for business solutions. The services are deployed and accessed through SOA infrastructures. They are governed and managed by SOA principles and management systems [244][62][63][201][204]. The ESOA brings forth the agility aspect to enterprise architectures, allowing enterprises to deal with system changes using a configuration mediation layer, rather than constantly having to redevelop these systems. However, ESOA introduces new challenges and issues to enterprise architecture because of its following characteristics:

- The enterprise owns the data center with ESOA services but the infrastructure is not dynamic enough to support auto scaling and elastic load balancing [10].
- The enterprise architecture is built behind firewalls.
- Resources are dedicated to each workload.
- Resources are shared only within the enterprise.

Figure 5.1 shows a traditional ESOA data center in which there are three layered infrastructures:

- Web server infrastructure;
- Enterprise application server and service infrastructure which includes application database and SOA services, application monitors and SOA application management;
- Enterprise information storage and business service infrastructure.

All enterprise services are operating behind firewalls.

Building a data center to support ESOA architecture is expensive and it is not possible for some small to medium enterprises. For large enterprises, it is not possible to complete some complex business processes, such as online shopping and shipping, without third party services. Moreover, many server resources in a large data center are idle or passive, such as during non-peak time periods, since the plan of resources is based on the highest volume of workload. Thus, resources are wasted resulting in increasing cost of resources and operations. Many enterprises view SOA as something that only occurs within firewall. The ESOA faces new challenges from enterprises – reducing complexity as well as cost and increasing capacity, flexibility as well as agility. Leveraging cloud computing to support a new paradigm of distributed computing for enterprises brings forth many new ideas, concepts, solutions, and principles to enterprise architecture and ESOA. Originally, cloud computing evolved from web computing (such as Web 2.0 [83]), service-oriented computing [213][214][244][204][63], grid computing [246], utility computing [35], and other technologies, including virtualization [95] and virtual applications. Cloud computing is about moving services, computation and/or data off-site to an internal or external, location-transparent, centralized facility or contractor for cost and business advantages. By making services and data available in the cloud, it can be more easily and ubiquitously accessed, often at much lower cost, increasing its value by enabling opportunities for enhanced collaboration, integration, and analysis on a shared common platform [52]. The framework is applied to model and analyze a new architectural style, Enterprise Cloud Service Architecture (ECSA), which is a hybrid of the ESOA and ECC styles.

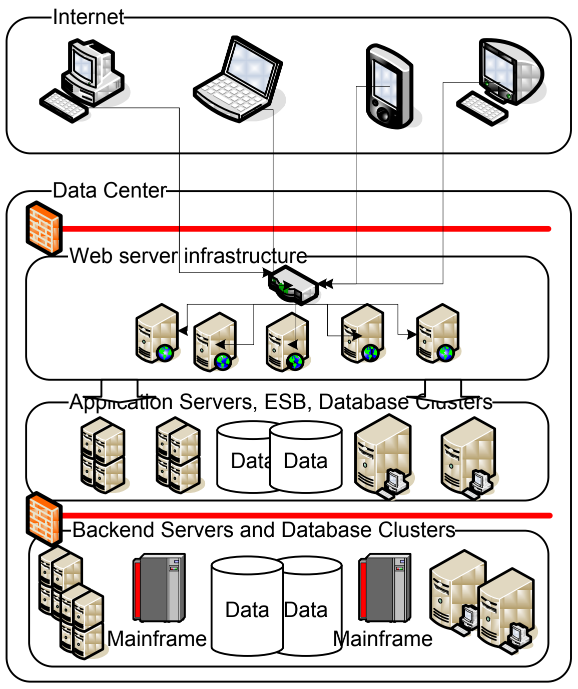

Internet
Data Center
Web server infrastructure
Application Servers, ESB, Database Clusters
Data
Data
Backend Servers and Database Clusters
Mainframe
Data
Data
Mainframe

Figure 5.1. Enterprise SOA Data Center

The rest of this chapter is organized as follows: The next section defines ECSA based on the framework in Chapter 3. Section 5.3 specifies ECSA and defines its quality ontology.

5.2 ECSA Ontology

This section defines the new ECSA style framework for modeling enterprise service-*oriented architecture* [201][204] and the ontology-based framework for modeling EAS styles presented in Chapter 3. Before defining the new style, we define

$$Cloud = \{\text{public cloud}\} \cup \{\text{private cloud}\}, \tag{5.1}$$

in which the public cloud, such as Google cloud [236] and Amazon cloud (Amazon Web Services - EC2 and S3) [173], is typically on the Internet or off-premise. The private cloud, such as cloud-enabled data center, is typically located on-premise.

Firstly, for some enterprises with both ESOA systems and cloud systems for serving different businesses and customers, we define architectural style ECSA as a combination of ESOA and ECC based on (3.17). Secondly, for some enterprises with only service-oriented private cloud systems, we discuss its refinement form in terms of (3.18).

$$ECSA = \langle \Sigma_{ECSA}, \Phi_{ECSA} \rangle, \tag{5.2}$$

in which

$$\Phi_{ECSA} = \Phi_{ESOA} \cup \Phi_{ECC}, \tag{5.3}$$

includes all concept descriptions for the concepts in Σ_{ECSA} where

$$\begin{aligned} \Sigma_{ECSA} &= \Sigma_{ESOA} \cup \Sigma_{ECC} \\ &= \langle S, C, D, SI, SM, SP, SQ, SD \rangle , \end{aligned} \tag{5.4}$$

where

$$S = S^{I} \cup S^{II} \cup S^{III}, \tag{5.5}$$

in which

S^{I} ={s | s is a service not on cloud},

S^{II} = {s| s is a service on private cloud},

S^{III} = {s | s is a service on public cloud}.

Let us define

S^{IV} = {s | s is a service on hybrid cloud},

S^{V} = {s | s is a service on community cloud}

where $S^{IV} \subseteq S^{II} \cup S^{III}$ and $S^{V} \subset S^{III}$.

$$C = C^{I} \cup C^{II}, \tag{5.6}$$

in which

C^{I} ={c | c is a non-cloud service consumer}

C^{II} ={c | c is a cloud service consumer}

$$D = D^{I} \cup D^{II}, \tag{5.7}$$

in which

D^{I} = {d | d is an SOA data element}

D^{II} = {d | d is a cloud data element}

$$SI = SI^{I} \cup SI^{II}, \tag{5.8}$$

in which

SI^{I} ={r | r is an SOA infrastructure}

SI^{II} ={r | r is a cloud infrastructure}

$$SM = SM^{I} \cup SM^{II}, \tag{5.9}$$

in which

SM^{I} ={m | m is an SOA management}

SM^{II} ={m | m is a cloud management}

$SP = SP^{I} \cup SP^{II}$, (5.10)

in which

SP^{I} ={p | p is an SOA process}

SP^{II} ={p | p is a cloud process}

$SQ = SQ^{I} \cup SQ^{II}$, (5.11)

in which

SQ^{I} ={q | q is an SOA quality attribute}

SQ^{II} ={q | q is a cloud quality attribute}

$SD = SD^{I} \cup SD^{II} \cup SD^{III}$, (5.12)

in which

SD^{I} ={d | d is a building element of development}

SD^{II} ={d | d is a service deployment type}

SD^{III} ={d | d is a service delivery model}

From the definitions of each element of (5.4), we can see that the ECSA combines both the ESOA and the ECC styles. The advantages of the combined ontology are (1) it provides more concepts and descriptions and (2) it covers broader architectural decision choices. For example, if S^{II}, S^{III}, C^{II}, D^{II}, SI^{II}, SM^{II}, SQ^{II}, SD are empty, then the style is equal to ESOA. (3) It can be regarded as a top level style and easy to extend to different sub-styles, such as ECSA-PrC (Enterprise Private Cloud Service Architecture). The disadvantages of the combination ontology are (1) concept overlapping, such as $SI^{I} \cap SI^{II} \neq ø$ and (2) description overlapping, since there are a lot of commonalities between ESOA and ECC. Figure 5.2 depicts the relationship of all parts in (5.4). In the following subsections, we will describe each element defined in formulae (5.4) to (5.12) in detail formally [200] and informally. Since ESOA has been described in Chapter 4, this chapter focuses on describing the new concepts in ECC and the relationship between ECC and ESOA.

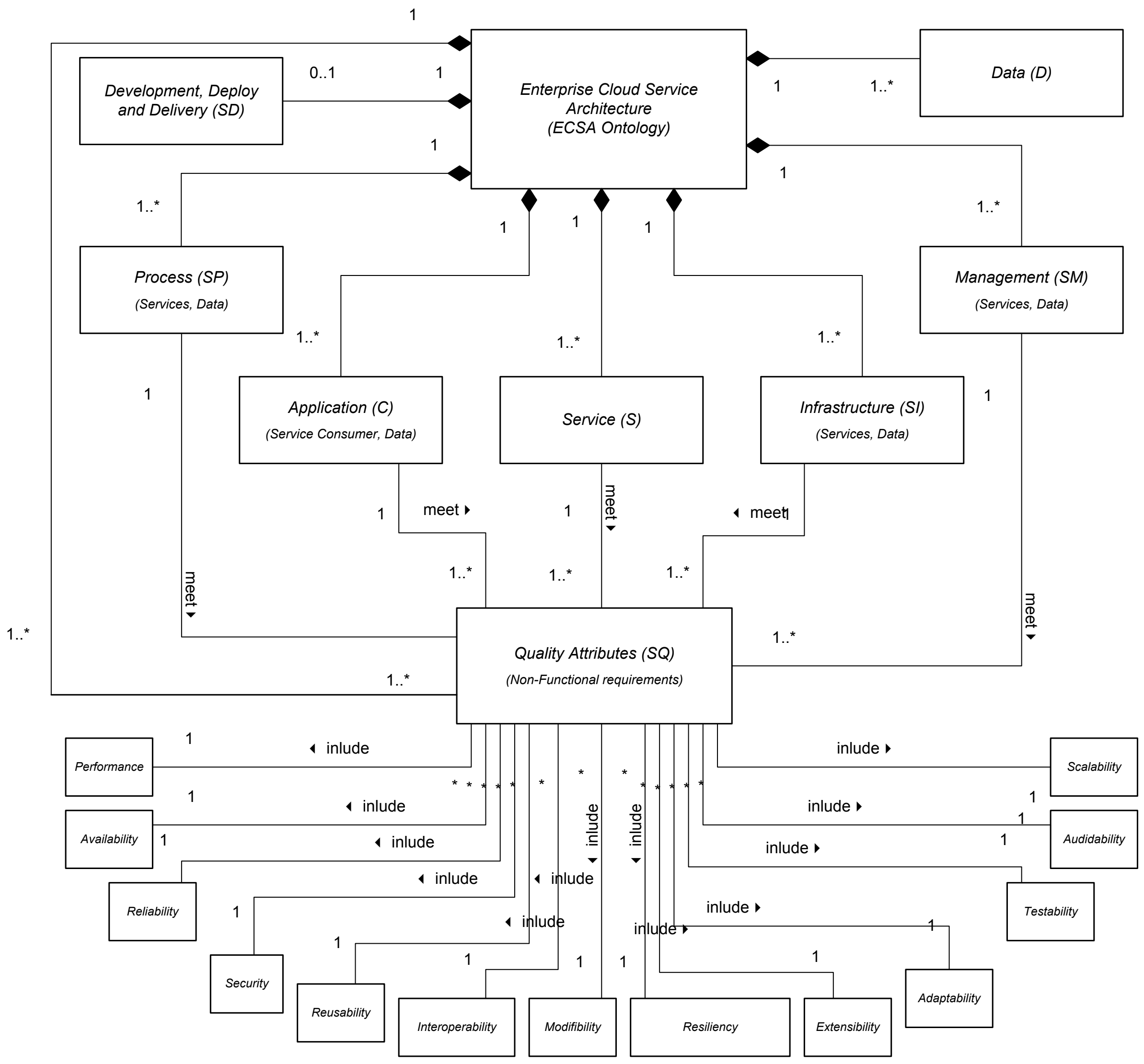

Enterprise Cloud Service Architecture (ECSA Ontology)
Development, Deploy and Delivery (SD)
Data (D)
Process (SP) (Services, Data)
Management (SM) (Services, Data)
Application (C) (Service Consumer, Data)
Service (S)
Infrastructure (SI) (Services, Data)
Quality Attributes (SQ) (Non-Functional requirements)
meet
inlude
Performance
Availability
Reliability
Security
Reusability
Interoperability
Modifibility
Resiliency
Extensibility
Adaptability
Testability
Audidability
Scalability
1
0..1
1..*
*

Figure 5.2. ECSA Domain Ontology

5.3 Specifying ECSA

In the section, ECSA is specified in detail formally and informally. Descriptions of each part in (5.4) focus on ECC and the relationship between ESOA and ECC.

5.3.1 A 3D Model of Cloud Services

Compared with the enterprise *service-oriented* ontology in (4.2), only one new element SD is added into the ECSA ontology (5.4). We call SD as the 3D model of Cloud Services. The 3D model with new concepts distinguishes between traditional ESOA services and cloud services as well as between ESOA style and ECSA style. In the 3D model, SD^{I} is a set of building blocks and tools for cloud service and enterprise application development, so it is the basis for developing the services in {PaaS} = {Platform as a Service}. SD^{II} is a set of cloud service deployment types:

SD^{II} ={PrC, PuC, VPC, CoC, HyC}.

Table 5.1. Deployment Types of Cloud Services

Deploy Type	**Description**
PrC	Private Cloud [125]
PuC	Public Cloud [125]
VPC	Virtual Private Cloud [225]
CoC	Community Cloud [125]
HyC	Hybrid Cloud [125]

Table 5.1 provides a description of each type where the CoC can be managed by the community or third party and may be on-premise or off-premise. The VPC was first created by Amazon [225]. It is a private cloud hosted in a public cloud through VPN network. The service consumers (SC's) resources are isolated in the public cloud, which provides an online virtual data center to

SC. The CoC infrastructure is shared by a community – several enterprises or organizations which have the same concerns, such as mission, security, and policy requirements.

SD^{III} is a set of delivery modes of cloud services as follows:

SD^{III} ={SaaS, PaaS, IaaS, IMaaS, IRaaS, XaaS}.

They are described in the following Table 5.2:

Table 5.2. Delivery Modes of Cloud Services

Delivery Mode	**Description**	**Resource sharing**
SaaS	Software as a Service [173]	Sharing software
PaaS	Platform as a Service [125]	Sharing platform
IaaS	Infrastructure as a Service [125]	Sharing infrastructure
IMaaS	Information as a Service [125]	Sharing information
IRaaS	Integration as a Service [125]	Sharing integration
XaaS	Other cloud service delivery model [125]	Sharing other resources

5.3.2 Services and Cloud Services

We have formally and informally specified services as self-contained software abstractions of business, technical functionality, or infrastructure management, defined by a well-defined interface [62]. We define the kind of enterprise services as functional services which serve business for completing certain operations, such as shopping transaction web service and hotel reservation web service. They include composed and process services, such as workflow services. If a functional service s is not exposed to the Internet (out of enterprise firewall) or it cannot be accessed from the Internet, then $s \in S^{I}$. In this dissertation, we focus on the managed services (or enterprise services) on the cloud. We define an Enterprise Cloud Service (ECS):

Enterprise Cloud Service (ECS) is a specific managed service with Service Level Agreement (SLA), elasticity/dynamism, accountability/utility, loosely-coupled, which can be accessed and delivered from the Internet.

If s is an ECS within the enterprise internal network, then $s \in S^{II}$ and s is called a private cloud service.

If s is an ECS in an enterprise cloud service provider network, then $s \in S^{III}$ and s is called a public cloud service.

Cloud computing extends the ESOA service concept and capacity to a broader area in two aspects – the vertical and horizontal views as shown in Figure 5.3.

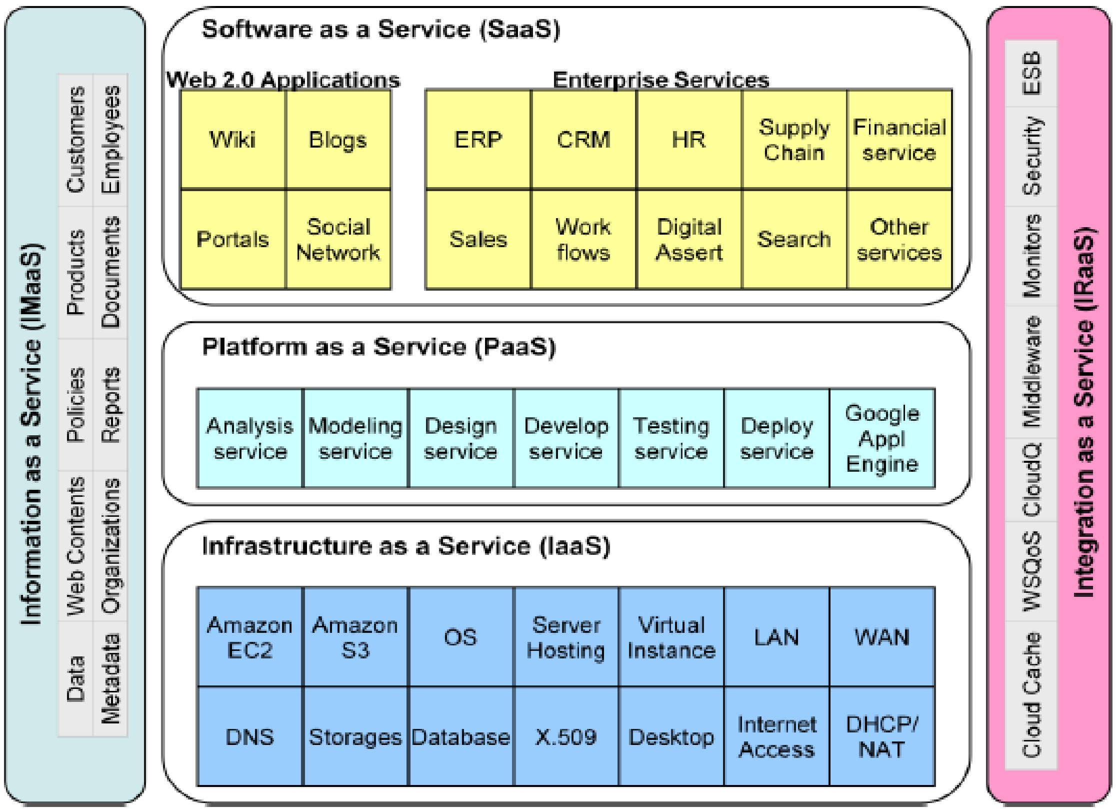


Figure 5.3. View of Cloud Services

We define

S_{ESOA}={s | s is a traditional SOA service}

and

S_{Cloud}={s | s is a *Cloud* service}

$= S_{Cloud}{}^{I} \cup S_{Cloud}{}^{II}$

in which $S_{Cloud}{}^{I}$= {SaaS} ∪ {PaaS} ∪ {IaaS}, which includes three kinds of basic cloud services; $S_{Cloud}{}^{II}$= {IMaaS} ∪ {IRaaS} ∪ {XaaS}, which includes other types of cloud services.

Thus, $S_{ESOA} \cap S_{Cloud} \neq$ Ø. If a service $s \in S_{ESOA} \cap S_{Cloud}$, then $s \in S^{II}$.

In the ESOA ontology, service is a primary type. However, if the cloud service $s \notin S_{ESOA}$, such as SaaS, PaaS, IaaS, and s is a composed service type, then we call it ECSType, such as PaaSType or IaaSType or SaaSType. Therefore, we can define its ontology based on our framework:

$$\text{ECSType} = \langle \Sigma_{ECS}, \Phi_{ECS} \rangle, \quad (5.13)$$

where Σ_{ECS} includes concepts and roles of ECS and Φ_{ECC} defines all concept descriptions for Σ_{ECS}.

To specify an *ECSType*, let us define the following sets of types of properties:

- Cloud Service Interface Type
  I_{type}={User Interaction Interface, Web Service Interface, REST Interface, Web Application Interface, Event Interfaces}
- Cloud Service Access Type
  A_{type}={a | a is a client access protocol method},

  such as Web User Interaction (HTTP), Web Service API (SOAP), REST API (HTTP), Web Application API, Event Trigger, distributed devices (wireless devices).
- Cloud Service Provisioning Type
  P_{type}={Applications, Business Operations, Resources, Information, Platform, Integration}
- Cloud Service Control/Ownership Type
  O_{type}={O_{own}, $O_{thirdparty}$}, in which

O_{own} = Buy/lease and Own, $O_{thirdparty}$ = Owned by public cloud provider and pay-as-you-go.

Now, we can specify Σ_{ECS} as an 8-tuple:

$$\Sigma_{ECS} = \langle I_{ECS}, A_{ECS}, P_{ECS}, O_{ECS}, SD^{II}_{ECS}, S^{III}_{ECS}, Policy_{ECS}, SQ_{ECS} \rangle, \tag{5.14}$$

where $I_{ECS} \subset I_{type}$, $A_{ECS} \subset A_{type}$, $P_{ECS} \in P_{type}$, $O_{ECS} \in O_{type}$, $SD^{II}_{ECS} \in SD^{II}$, $SD^{III}_{ECS} \in SD^{III}$,

$SQ_{ECS} = QoS_{ECS} + SLA_{ECS} \subset SQ$

in which QoS_{ECS} is the *ECS* Quality of Service and SLA_{ECS} is the *ECS* Service Level Agreement between *ECS* provider and its consumer. For instance, the Amazon EC2 cloud service [173] can be specified as

$$EC2_{Amazon} = \langle I_{EC2}, A_{EC2}, P_{EC2}, O_{EC2}, SD^{II}_{EC2}, S^{III}_{EC2}, Policy_{EC2}, SQ_{EC2} \rangle$$

where I_{EC2} = {Web service interface, REST interface},

A_{EC2} ={Web service API, REST API}, P_{EC2} = resources,

$O_{EC2} = O_{thirdparty}$ (owned by Amazon), SD^{II}_{EC2} =PuC, SD^{III}_{EC2} =IaaS, ($Policy_{EC2}$, SQ_{EC2}) has two parts – the documentation can be found in [9] and runtime policy and SLA are managed by Amazon's runtime cloud management.

Using the style syntax defined in Section 3.1, ECSType can be defined as

ECSType $\subseteq$ ECSConfiguration (t)

ECSType $\equiv \exists$ hasPart.(ECSComponent $\bigcup$ ECSConnector $\bigcup$ Port $\bigcup$ Role $\bigcup$ QA)

For instance, in IaaSType, the ECSConfigration (t) = managed dynamic Infrastructure (t),

ECSComponent $\equiv$ ECSType $\bigcap \exists$ hasInterface.Port

ECSConnector $\equiv$ ECSType $\bigcap \exists$ hasEndpoint.Role

In IaaSType, the major components include web service, service provisioning service, monitoring agent, and resource virtual instance. The connector is the glue between web service and virtual server instance through SOAP or REST protocol. The Amazon EC2 is one instance of the IaaSType.

5.3.3 Cloud Service Consumers

We have specified the ESOA service consumers C^{I} and part of the consumers of private cloud services in [204]. The part of ESOA service consumers are also part of consumers of private cloud services. In this chapter, we focus on specifying the enterprise cloud service consumers C^{II}. In Figure 5.4, we show that there are four kinds of enterprises with different ECSA architectural styles:

- Enterprise A has no data center and it is a consumer of public cloud of the provider Enterprise B. Most small to medium enterprises typically are or will become this kind of enterprises.
- Enterprise B is a public cloud provider which provides public cloud services, such as Amazon cloud [173], Google cloud [236], Saleforce cloud [173], IBM cloud center [97], and Microsoft Azure cloud [142].
- Enterprise C has data center with private cloud services whose consumers are cloud applications accessed by internal customers, such as registered users, employees and partners. The private cloud services can be the consumers of other public cloud services in SEDC (somebody else's data center).
- Enterprise D has multiple data centers and hybrid clouds. The consumers of its public cloud services can be private cloud inside the enterprise, and internal and external cloud

applications accessed by external clients that include external end-users and cloud applications in other enterprises. The consumers of its private cloud services can be internal applications accessed by internal clients that include internal end-users and the public cloud services within the collocation. Most large enterprises are or will become this kind of enterprises.

- The cloud service consumers also depend on the type of the cloud service. If the *ECS* is in {PaaS}, such as Google App Engine, then web software developers, IT managers and application system administrators are the consumers of *ECS*. If the *ECS* is in {IaaS}, then the system and database administrators are its consumers.

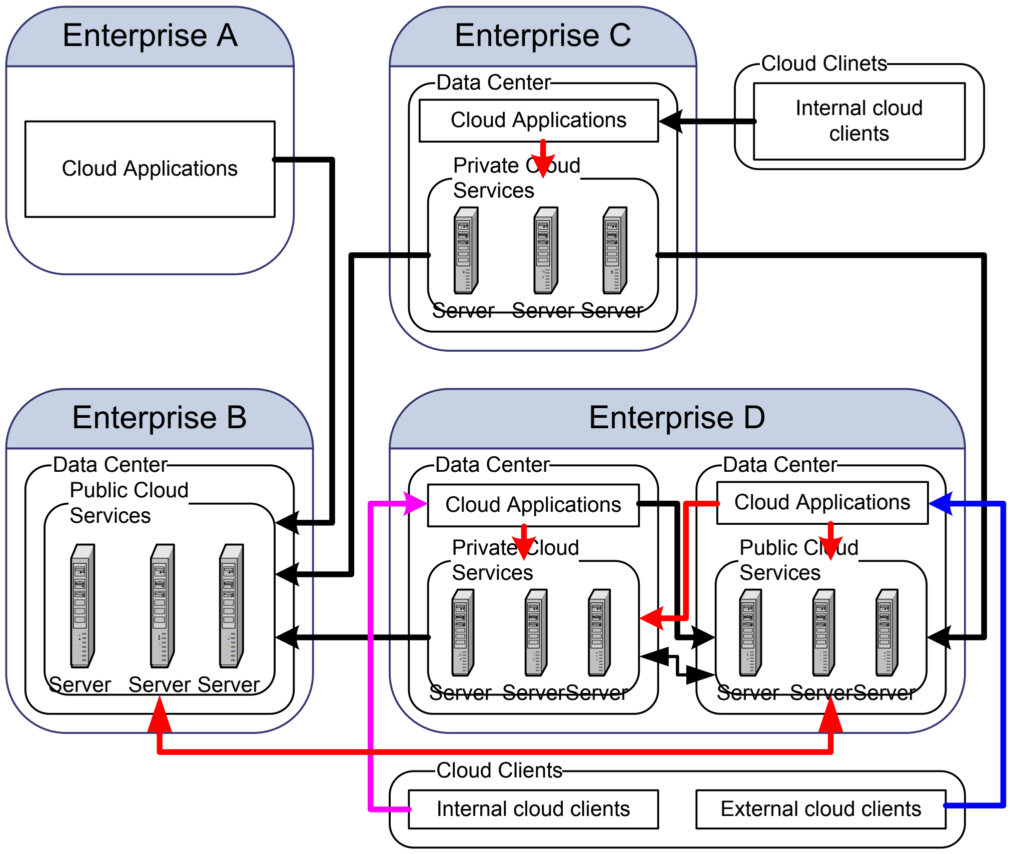


Figure 5.4. Cloud Service Consumers

Specifically, the public or private cloud service consumers have the following characteristics:

- Self-service: Users can access services they provide or directly procure services in the cloud. Users also manage and monitor cloud services from self-service portals.
- Standard API for accessing cloud services.
- Rapid service provisioning.
- Pay-for-use.

ESCA service consumers are traded as one of primary component types of EAS ontology in Chapter 3.

5.3.4 SOA and Cloud Data

The set D in (5.7) consists of two sets of ECSA data elements which are used for building ECSA style enterprise architecture (EA). The SOA data set D^{I} has been specified in [201][204]. Part of the data and metadata in D^{I} are also used by cloud services, infrastructure and management, such as various resources and their profiles, basic infrastructure configuration data, and SOA metadata. However, cloud computing needs some cloud specific data and metadata in D^{II} as shown in Table 5.3.

5.3.5 SOA and Cloud Infrastructure

The traditional SOA infrastructure SI^{I} is the heart of ESOA. It is the bridge for the transformation between business and services. For the new ECSA style, the cloud infrastructure is added. It is easy to show that $SI^{I} \cap SI^{II} \neq \emptyset$ which means that the infrastructure of the new style is a hybrid of both the SOA and the cloud infrastructure styles. The traditional ESOA

Table 5.3. Cloud Data

Cloud Data	**Examples from public or private cloud**
Virtual resources	Virtual instance, virtual server, virtual OS, virtual network, virtual storage
Application metadata	Google App Engine application metadata
Cloud policy	Security policy, Routing policy, Privacy policy, Access policy (such as Amazon web services REST/SOAP access control policy)
Cloud SLA	Error rate, Monthly update percentage, Service credit, Region Unavailable
Utility model data	Pricing (such as EC2 high CPU on-demand instances – Medium UNIX \$0.20/per hour), Billing, Paying for what user used.
Virtualization metadata	The virtualization metadata contains all setup and configuration information required for the virtualization layer to establish a connection and it may also contain additional information to make some specific operations (examples of metadata are: server name, database name, user, password, translation fields, etc.). It is usually described by a XML schema and stored in metadata repositories or database.
Application network delivery metadata	It includes all setup and configuration information required for application delivery infrastructure, such as load balancing, acceleration, optimization, and security.
Infrastructure instance metadata	EC2 instance metadata [11]
Cloud configuration data and metadata	Types of resources (such as CPU, Storage, OS, Software, Monitoring), Types of instances (such as Amazon EC2 – Small, large and extra large instances, High-CPU medium and extra large instances)

infrastructure is not really dynamic and flexible enough. Therefore, it is not adaptable to today's on-demand business workloads and real-time B2B requirements. It also uses more resources and power in enterprise's data center. The cloud infrastructure SI^{II} is a dynamic IT infrastructure which consists of elastic web servers, elastic application servers, elastic MQ servers, and elastic database servers. It has the following three main characteristics:

- It supports elasticity and dynamism – automatic scalability and load-balancing, failover in terms of virtualization [35][52][86] or other technologies [236].
- It supports resource usage accountability – utility model [35][245].
- It can be a part of cloud service, such as PaaS type services (Google App Engine), or it can be a cloud service, such as IaaS type service (Amazon EC2).

Therefore, the cloud infrastructure brings cost-effective operations and elasticity to current SOA infrastructures. The traditional SOA infrastructure is now refined with cloud infrastructure's dynamism. We can specify a dynamic infrastructure ontology based on the framework in Chapter 3. We assume a typical dynamic enterprise infrastructure Figure 5.5:

IComponents = {INET, DLB, VWEB, PWEB, VAS, PAS, VST, PST, VNET, PNET, IMS, ISS}

in which

INET = Internet;

DLB = Dynamic Load Balancer;

VWEB = Virtual Web Server;

PWEB = Physical Web Server;

VAS = Virtual Application Server;

PAS = Application Server;

VST = Virtual Storage;

PST = Storage;

VNET = Virtual Network;

PNET = Physical Network;

IMS=Infrastructure Management Service;

ISS=Infrastructure Security Service

IConnectors={INET-DLB, DLB-VWEB, VWEB-PWEB, PWEB-VNET, PWEB-VST, PWEB-VAS,VAS-PAS, PAS-VNET, PAS-VST, VST-PST, PST-VNET, VNET-PNET, IMS-DLB,IMS-VWEB, IMS-PWEB, IMS-VAS, IMS-PAS, IMS-VST, IMS-PST, IMS-VNET, IMS-PNET, ISS-

DLB, ISS-VWEB, ISS-PWEB, ISS-VAS, ISS-PAS, ISS-VST, ISS-PST, ISS-VNET, ISS-PNET}

Now we can define a dynamic infrastructure by using ACME-like language:

```
Infrastructure(t) DynInfrastructure = {
   IComponent Type DLB = {
      Ports = {In, Out}};
   IComponent Type VWEB = {
      hasVirtualInterface = true
      Ports = {In, Out}};
   IComponent Type PWEB = {
     hasPhysicalInterface = true
     Ports = {In, Out}};
   IComponent Type VAS = {
     hasVirtualInterface = true
     Ports = {In, Out}};
   IComponent Type PAS = {
     hasPhysicalInterface = true
     Ports = {In, Out}};
   IComponent Type VST = {
     hasVirtualInterface = true
     Ports = {In, Out}};
   IComponent Type PST = {
     hasPhysicalInterface = true
     Ports = {In, Out}};
IComponent Type VNET = {
     hasVirtualInterface = true
     Ports = {In, Out}};
IComponent Type PNET = {
    hasPhysicalInterface = true
    Ports = {In, Out}};
IConnector Type INET-DLB = {
    Roles = {request, dynamicLoadReq}};
IConnector Type DLB-VWEB = {
    Roles = {dynamicLoadReq, provideWeb}};
IConnector Type VWEB-VNET = {
    Roles = {routRequest, provideRout}};
IConnector Type VWEB-PWEB = {
    Roles = {requestWebExec, provideWebExec}};
IConnector Type PWEB-VNET = {
    Roles = {routRequest, provideRout}};
IConnector Type PWEB-VAS = {
    Roles = {reqAppProcess, provideAppProcess}};
IConnector Type VAS-VNET = {
    Roles = { routRequest, provideRout }};
```

```
IConnector Type VAS-PAS = {
    Roles = {requestTransExec, provideTransExec}};
IConnector Type PAS-VNET = {
    Roles = {routRequest, provideRout}};
IConnector Type PAS-VST = {
    Roles = {reqDataAccess, provideDataAccess}};
IConnector Type VST-VNET = {
    Roles = {routPST, provideRout }};
IConnector Type VST-PST = {
    Roles = {reqAccessExec, provideAccessExec}};
IConnector Type VNET-PNET = {
    Roles = {reqRoutExec, provideRoutExec}};
IConstraint Type scalability = {
    dynamically scale out and scale down, on-demand};
IConstraint Type availability = {
    SLA-defined, high fault-tolerance};
IConstraint Type agility = {
    resilient computing, dynamic reconfiguration};
IConstraint Type security = {
    End-to-End security checking};
Ownership Type OwnedbyEnterprise = {
    IaaS in PvC enterprise's datacenetr};
```

```
  Ownership Type OwnedbyVendor = {
      IaaS in PuC Vendor's datacenter};
}
```

In the specification, the management components, End-to-End Infrastructure management service and End-to-End infrastructure security service, are ignored. They are discussed in the next section. Figure 5.5 is the high-level graphic description of a typical enterprise layered dynamic infrastructure.

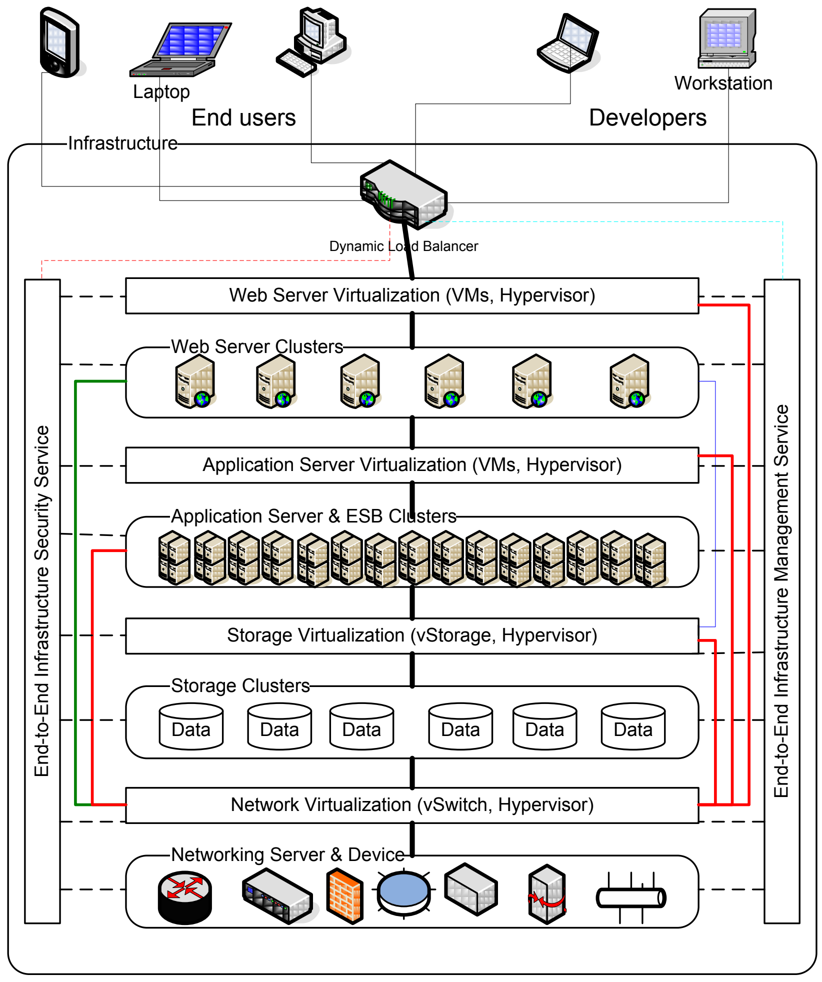

Laptop
End users
Workstation
Developers
Infrastructure
Dynamic Load Balancer
Web Server Virtualization (VMs, Hypervisor)
Web Server Clusters
Application Server Virtualization (VMs, Hypervisor)
Application Server & ESB Clusters
Storage Virtualization (vStorage, Hypervisor)
Storage Clusters
Data
Data
Data
Data
Data
Data
Network Virtualization (vSwitch, Hypervisor)
Networking Server & Device
End-to-End Infrastructure Security Service
End-to-End Infrastructure Management Service

Figure 5.5. Dynamic Enterprise Infrastructure

5.3.6 SOA and Cloud Management

Cloud computing is changing the landscape of ESOA and brings forth new types of services and dynamic infrastructures into ESOA. An enterprise architecture needs SOA to achieve better quality by leveraging cloud computing providers [125]. The relatively mature SOA management or governance SM^{I} is the foundation of cloud management SM^{II}. It is easy to show that $SM^{I} \cap SM^{II} \neq Ø$. The SOA management we have specified in Chapter 4, such as network and application monitoring, identity management, policy enforcement, service-level agreement management, and service lifecycle management in SM^{I}, are very important for cloud computing. Thus, they are also in SM^{II}. However, cloud computing extends the SOA management to a new level from two perspectives, namely, enhancing SOA managements and adding some new cloud specific managements, since:

- Cloud systems are more dynamic and mostly real-time with automatic runtime governance compared to services infrastructure.
- Cloud systems request highly automatic policy and SLA management at runtime.
- Cloud systems request an automatic service provisioning management for their utility model.
- Cloud systems need new identity management for cloud service security and trust, such as the Amazon cloud security process [12].

We specify the refinement ontology of cloud service management with SOA management in terms of ACME-like language and the framework defined in Chapter 3. Assume

MComponents = {MS, SS, RS, SLM, LM, IM, AM} in which

MS = Monitoring service

SS = Security management service

RS = Resource management service

SLM = Service Level Management service [206]

LM = service lifecycle management service

IM = Infrastructure management service

AM=Account management service which includes cloud service consumer's utility billing management.

MConnectors ={MS-IC, MS-AC, MS-SC, MS-SPR, SS-IC, SS-AC, SS-SC, SS-SPR, RS-IC, SC-SLM, SLM-SPR, MS-SLM, SLM-RS, LM-SPR, IM-IC} in which SC = Service consumer, SPR = Service Provider and AC = Application Component are management service consumers, IC = Infrastructure Component which include management service consumer and managed resources.

```
Management(t) CloudServiceManagement = {
    MComponent Type MS = {
        hasInterface = true
        Ports = {In, Out}};
    MComponent Type SS = {
        hasInterface = true
        Ports = {In, Out}};
    MComponent Type RS = {
        hasInterface = true
        Ports = {In, Out}};
    MComponent Type SLM = {
```

```
   hasInterface = true
   Ports = {In, Out}};
MComponent Type LM = {
   hasInterface = true
   Ports = {In, Out}};
MComponent Type IM = {
   hasInterface = true
   Ports = {In, Out}};
MComponent Type AM = {
   hasInterface = true
   Ports = {In, Out}};
MConnector Type MS-IC = {
   Roles = {monitoring, runtime}};
MConnector Type MS-AC = {
   Roles = {monitoring, runtime}};
MConnector Type MS-SC = {
   Roles = {monitoring, runtime}};
MConnector Type MS-SPR = {
   Roles = {monitoring, runtime}};
MConnector Type MS-SLM = {
   Roles = {report, negotiatingOffering}};
MConnector Type SS-IC = {
   Roles = {protect, resources}};
MConnector Type SS-AC = {
   Roles = {secure, access}};
MConnector Type SS-SPR = {
   Roles = {secure, access}};
MConnector Type SS-SC = {
   Roles = {secure, access}};
MConnector Type SC-SLM = {
   Roles = {reqService, claimQoS}};
MConnector Type SLM-SPR = {
   Roles = {offerService, provideService}};
MConnector Type SLM-RS = {
   Roles = {reportResource, mgrResources}};
MConnector Type SLM-RS = {
   Roles = {reportResource, mgrResources}};
MConnector Type RS-IC = {
   Roles = {mgrResources, dynReconfig}};
MConnector Type LM-SPR = {
   Roles = {mgrLifeCycle, provideDynService}};
MConnector Type IM-IC = {
   Roles= {mgrInfrastructure, glueSCAndServices}
Constraint Type CManagedByEnterprise = {
        componentsOwnedByEnterprise = true}
    Constraint Type CManagedByVendor = {
        componentsOwnedByVendor = true}
    Constraint Type CManagedByBoth = {
        componentsOwnedByBoth = true}};
}
```

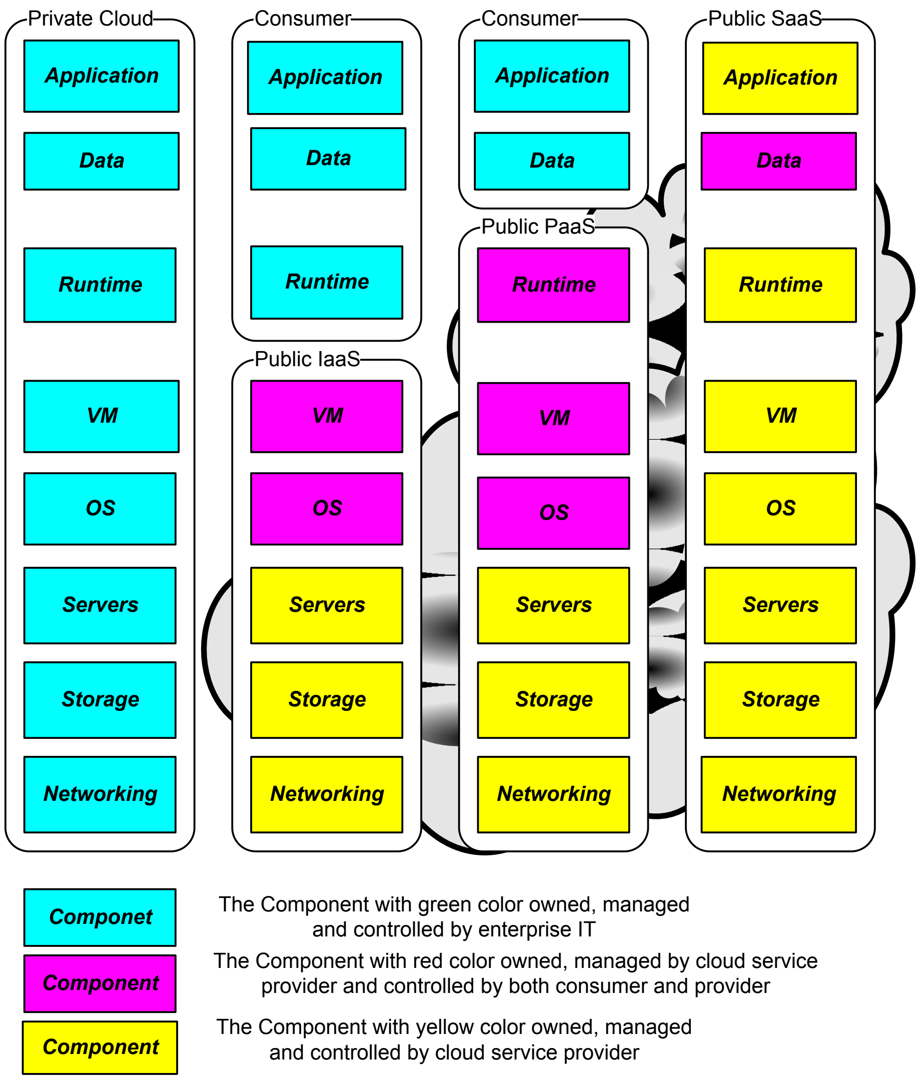


Figure 5.6. Boundaries of Components

Figure 5.6 roughly describes the boundaries of components in different types of cloud services. From the figure, we can see a remarkable difference between traditional SOA management and cloud management. Traditional SOA management manages all components in its own data center, but cloud management manages different sub-sets of components by different component ownership.

5.3.7 SOA and Cloud Process

One of the important parts of the ESOA style is its set of SOA processes. The SOA process or workflow is an abstraction of Business Process Management (BPM). Each process is composed of multiple services in orchestration and/or choreography for completing a whole or partial business process or task. The traditional SOA process can be executed by using an ESOA infrastructure with process engine in the internal network of an enterprise. However, the traditional SOA processes face many challenges and issues: real-time high performance (such as automated trading), on-demand scalability, large payloads (10+ MB), memory constraints, and high availability and reliability. In a distributed SOA environment of an enterprise, the bottlenecks tend to occur in one or more of the following three places:

- Shared intermediary services;
- The services themselves;
- SOA infrastructure operations.

In most cases, the scalability bottlenecks across all these SOA parts in workflow/process are caused when disk I/O, memory, or CPU saturation levels are reached. Moreover, the cluster technology, adopted by traditional SOA, can provide higher availability. However, it depends on static partitioning, where a single backup server is pre-assigned to service requests from a failing server. The grid-enabled SOA provides a way to improve the performance, scalability, and availability of SOA processes. Cloud computing shares the same goal as grid computing, namely, to allow service consumers to obtain computing resources on-demand. However, cloud computing is a new style of distributed computing, which introduces many new architectural styles and technologies to SOA. Compared with grid computing, there are four aspects in which cloud computing differs from grid computing [72]:

- It is massively scalable;
- It can be encapsulated as an abstract entity that delivers different levels of services to the customers outside of the Cloud;
- It is driven by economies of scale;
- The services can be dynamically configured through virtualization and other approaches and delivered on-demand.

We describe the ECSA style Business Process Management (BPM) process based on the framework in Chapter 3 and ACME-like language as follows. Assume in Figure 5.7

PComponent = {BPMB, BPMM, BPME, SPPR, SP, BPMS}, in which

BPMB=BPM Process Buider

BPMM=BPM Process Managment

BPME=BPM Process Engine (Runtime)

SPPR=SOA Process Provider

SP=SOA BPM Process

BPMS=BPM Process Storage

PConnector = {SC-BPMM, BPMB-BPMM, BPMM-BPME, BPMM-BPMS, BPMM-SP, BPME-BPMS, BPME-SPPR, SP-SIN, SP-SOU}, in which

SC=Service Consumer

SIN=Services in enterprise datacenter (on-promise)

SOU=Services in cloud provider's datacenter

```
Process(t) CloudServiceProcess = {
  IComponent Type BPMB = {
     hasInterface = true
     Ports = {In, Out}};
  IComponent Type BPMM = {
     hasInterface = true
     Ports = {In, Out}};
  IComponent Type BPME = {
     hasInterface = true
     Ports = {In, Out}};
  IComponent Type SPPR = {
     hasInterface = true
     Ports = {In, Out}};
  IComponent Type SP = {
     hasInterface = true
     Ports = {In, Out}};
  PConnector Type SC-BPMM = {
```

Roles = {requestProcess, manageRequest}};

PConnector Type BPMB-BPMM = {

Roles = {buildProcess, registerProcess}};

PConnector Type BPMM-BPME = {

Roles = {sendRequest, runControlProcess}};

PConnector Type BPMM-BPMS = {

Roles = {requestInfo, storeProcessInfo}};

PConnector Type BPME-BPMS = {

Roles = {accessInfo, storeProcessInfo}};

PConnector Type BPMM-BPMS = {

Roles = {accessInfo, storeProcessInfo}};

PConnector Type BPMM-SPPR = {

Roles = {monitorControlSP, provideSP}};

PConnector Type SP-SIN = {

Roles = {executeSP, provideService}};

PConnector Type SP-SOU = {

Roles = {executeSP, provideCloudService}};

Ownership Type BPMOwnedbyEnterprise = {

SI=IaaS, BPMB and BPMM in PvC enterprise's Datacenter; BPMaaS $\in$ {PvC}};

Ownership Type SIOwnedbyVendor = {

SI=IaaS in PuC Vendor's datacenter; BPMB and

BPMM in PvC enterprise's datacenter; BPMaaS $\in$ {HyC} $\cap$ {IaaS}};

Ownership Type BPMOwnedbyVendor = {

SI=IaaS, BPMB and BPMM in PuC Vendor's datacenter;

BPMaaS $\in$ {PuC} $\cap$ {PaaS}};

}

Figure 5.7 depicts the typical topology of ECSA PConfiguration (t) virtually.

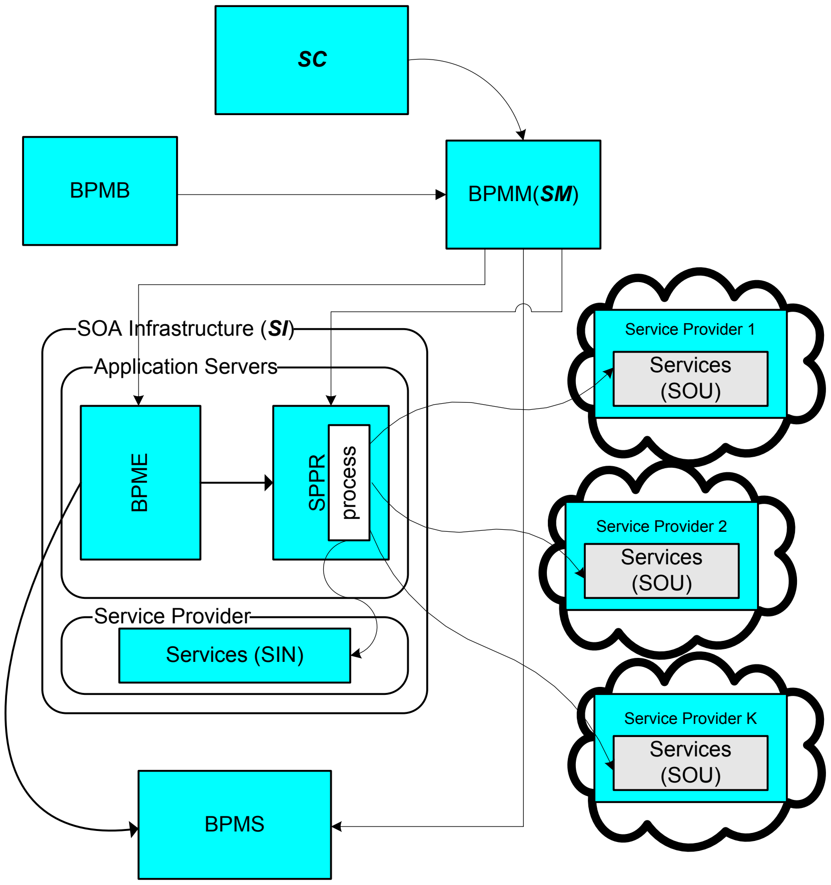


Figure 5.7. Typical Topology of ECSA Process

In Figure 5.7, the ***SI,*** BPMB, and BPMM could be provided by cloud service provider. For cloud, PConfiguration(t) is dynamic, in which dynamic process can be recomposed by choosing different service providers based on SLA [206]. Therefore service management is also reconfigured by selected different service providers, and MConfiguration(t) is also dynamically changed. [218] proposed a service-oriented dynamic reconfiguration framework for dependable distributed computing. [219] presented an approach of ontology-based dynamic process collaboration in SOA. Their work is of theoretical and practical significance for dynamic process management in ECSA.

5.3.8 Cloud Quality Attributes

The software architectural quality attributes [158] include not only the principles of system architecture design, but also the non-functional constraints of structure and behavior of any software architecture. Therefore, we include the architectural quality attributes as part of ECSA. We have defined common SOA quality attributes SQ^{I} of ESOA style in Chapter 4. They are also quality attributes of cloud computing, specifically, private clouds.Therefore, $SQ^{I} \cap SQ^{II} \neq Ø$. They both share many commonalities, such as performance, security, scalability, and availability. However:

(1) The quality attributes of SOA and public cloud have different degrees of maturity. In general, the maturity of cloud quality attributes is less than that of SOA quality attributes;

(2) The specifications of some of cloud quality attributes are different from traditional ESOA, such as elastic scalability; and

(3) SQ^{II} includes some cloud-specific quality attributes and properties of cloud services, such as cloud visibility and subscription.

We have not described ESOA quality ontology in Chapter 4. The next section defines and describes the ECSA quality ontology based on [167] and Chapter 3. It is an extension of the description of quality attributes in Chapter 4.

5.4 ECSA Quality Ontology

M. Klein and R. Kazman proposed an Attribute-Based Architectural Styles (ABAS) in [110], in which the architectural style's topology is specified and quality attribute response behavior is used as analysis and reasoning of the style's topology. C. Pahl, et al., introduced a quality ontology which extends the style ontology to capture a vocabulary of quality attributes (non-functional characteristics) and corresponding quality metrics [167][168]. In this dissertation, the quality ontology is defined as part of the style ontology in Chapter 3. The ECSA style's quality ontology [207] is specified based on our framework and (3.11) and (3.12):

$\text{CloudSQType} \subseteq \text{Performance} \cup \text{Reliability} \cup \text{Scalability} \cup \text{Reusability} \cup \text{Maintainability} \cup$
$\text{Security} \cup \text{Cost} \cup \text{Interoperability} \cup \text{Availability} \cup \text{Flexibility} \cup \text{Manageability} \cup \text{Agility} \cup$
$\text{Recoverability} \cup \text{Resiliency} \cup \text{Visibility} \cup \text{Accountability} \cup \text{Portability} \cup$
Compatibility

$\text{CloudSQ } (SQ^{II}) \equiv \exists \text{ hasTradeoff.(Performance} \cup \text{Reliability} \cup \text{Scalability} \cup \text{Reusability} \cup$
$\text{Maintainability} \cup \text{Security} \cup \text{Cost} \cup \text{Interoperability} \cup \text{Availability} \cup$
$\text{Flexibility} \cup \text{Manageability} \cup \text{Agility} \cup \text{Recoverability} \cup \text{Resiliency} \cup$
$\text{Visibility} \cup \text{Accountability} \cup \text{Portability} \cup \text{Compatibility)}$

We define and specify major quality attributes of cloud service systems in this section.

5.4.1 Cloud Performance (CSP)

The CSP is one of the most important runtime interaction behaviors of the cloud service architecture. Formally,

$\text{CSP} \equiv \exists \text{ hasMetric.(ResponseTime} \cup \text{Latency} \cup \text{Throughput)} \cap$

$\exists \text{ hasImpactFactor.(CloudServiceType} \cup \text{CloudServiceProvider} \cup \text{Availability} \cup$

$\text{Security} \cup \text{PayService})$

where

$$\text{ResponseTime} = RT(payload, bandwidth, roundtrips, RTT, T_S, T_C)$$

$$\approx \sum_{i=1}^{N} (\alpha_i \frac{payload_i}{bandwidth_i} + \beta_i RTT_i + T_S^i + T_C^i), \quad (5.15)$$

in which α, $\beta > 0$, N is the number of roundtrips between cloud service consumer and service providers; RTT_i is network Round Trip Time; T_S^i, T_C^i are cloud service computing time and client computing time at the *ith* application turn, respectively.

The cloud application may take multiple network round trips across different enterprise data centers through Internet and WAN (Wide Area Network), so latency is an important characteristic of cloud service computing. Latency impacts not only ResponseTime, but also Throughput. Knowing cloud latency greatly helps with architecture design and analysis.

$$\text{Latency} = L(L_{SI}, L_{SP}, L_{DT}, L_{SDP}, L_{LP}, L_{SC})$$

$$= \sum_{i}^{N} (a_i L_{SI}^i + b_i L_{SP}^i + c_i L_{DT}^i + d_i L_{SDP}^i + e_i L_{PL}^i + h_i L_{SC}^i), \quad (5.16)$$

where $a_{i,}, b_i, c_i, d_i, e_i, h_i > 0$, which are the *ith* cloud service provider and its environment related parameters, and L_{SI}^i is the *ith* cloud infrastructure latency which includes low level latency, such as OS, CPU, and Storage I/O latency, high level latency, such as VM, DNS, and Router/Switch latency, and networking latency, such as Internet and WAN latency. It can be formally described as

$$L_{SI}^i = L_{SI}^i\ (Distance_i,\ Hops_i,\ FML,\ BL,\ PL,\ LML,\ LSP)$$

where *$Distance_i$* is the geographical distance between cloud client and the *ith* cloud service; *$Hops_i$* is the set of “hops” in the network; *FML* is the first miles latency; *BL* is the backbone latency; *PL* is the peering latency and *LML* is the last miles latency. *LSP=LSP(LQ,LRMD,LIC,LID)* is the Latency of service provisioning (such as IaaS EC2) before service runtime execution, in which *LQ* is the latency from task request queuing [222]; *LRMD* is the latency from resource management decision program (reject task request because of insufficient capacity or schedule a job for the task request); *LIC* is the latency from creating instance (such as EC2 instance); *LID* is the latency from instance deployment. Figure 5.8 shows the cloud performance challenge, in which the public cloud datacenter is like Amazon Web Service (AWS), and IBM Cloud Datacenter whereas the private cloud datacenter and end users are the consumers of public cloud services. Since public cloud services are located in cloud provider’s datacenters, the *$Distance_i$* plays an important role in WAN latency: the larger the worse. For example, the load time from Bejing has more than two seconds latency than from New York on more than 10% uptime based on a report from BitCurrent.com [28]. Moreover, in (5.16), L_{SP}^{i} is the *ith* service processing latency; L_{DT}^{i} is the data transmission latency; L_{SDP}^{i} is the service dependency latency; L_{PL}^{i} is the propagation latency and L_{SC}^{i} is the service client latency which is coming from slow client processing.

The throughput describes the amount of tasks which can be performed over a given period of time. Therefore, throughput is another important performance metric. The maximum TCP/IP networking can be measured by the following formula [237]:

Throughput.Max.metrics =TWS/RTT, (5.17)

in which TWS is TCP Window Size and RTT is Round Trip Time. The cloud application throughput can be described by the following equation in terms of threads and latency:

$$\text{Throughput.metrics} = \frac{NumberofThreads}{APT + Latency}, \tag{5.18}$$

where APT is the cloud application processing time in seconds and Latency is the total latency in seconds. From (5.18), we see that Latency reduces Throughput and reducing Latency can increase Throughput.

Cloud Performance can be measured by several metrics which are based on distributed computing performance metrics. However, the CSP also has cloud factors which impact cloud performance:

Performance.CloudFactor1=CloudServiceType

{IaaS, PaaS, SaaS}

To describe the performance impact on the factor, we define a concept, cloud affinity level (CAL), as an indicator of a close degree of service consumer and other components in the enterprise architecture. The higher the CAL the lesser is the latency and the better is the performance. From Figure 5.6, we see that private cloud and SaaS have the highest CAL, IaaS has the lowest CAL. The BitCurrent report [28] shows that IaaS, such as Amazon EC2, may have higher latency than PaaS, such as Google AppEngine, or SaaS, such as Saleforce, in terms of the same resource.

Performance.CloudFactor2=CloudServiceProvider which can impact performance in two main aspects: (1) the geographic distance between cloud service consumer and the service provider datacenter; (2) cloud service provider's over or under subscription to resources, such as physical servers, VCPU, may impact the cloud performance.

Performance.CloudFactor3=Availability which can directly and indirectly impact performance. If services are not available (such as outages), then performance = zero. If partial services are not available and services failed over to other services in other data center, it will reduce cloud performance.

Performance.CloudFactor4=Security reduces performance normally.

Performance.CloudFactor5=PayService which impacts performance, since when a user pays more, the user can get more resources and services. For example, paying $0.17/per hour can get a medium CPU, but paying $0.68/per hour can get an extra large CPU from Amazon EC2 IaaS [9].

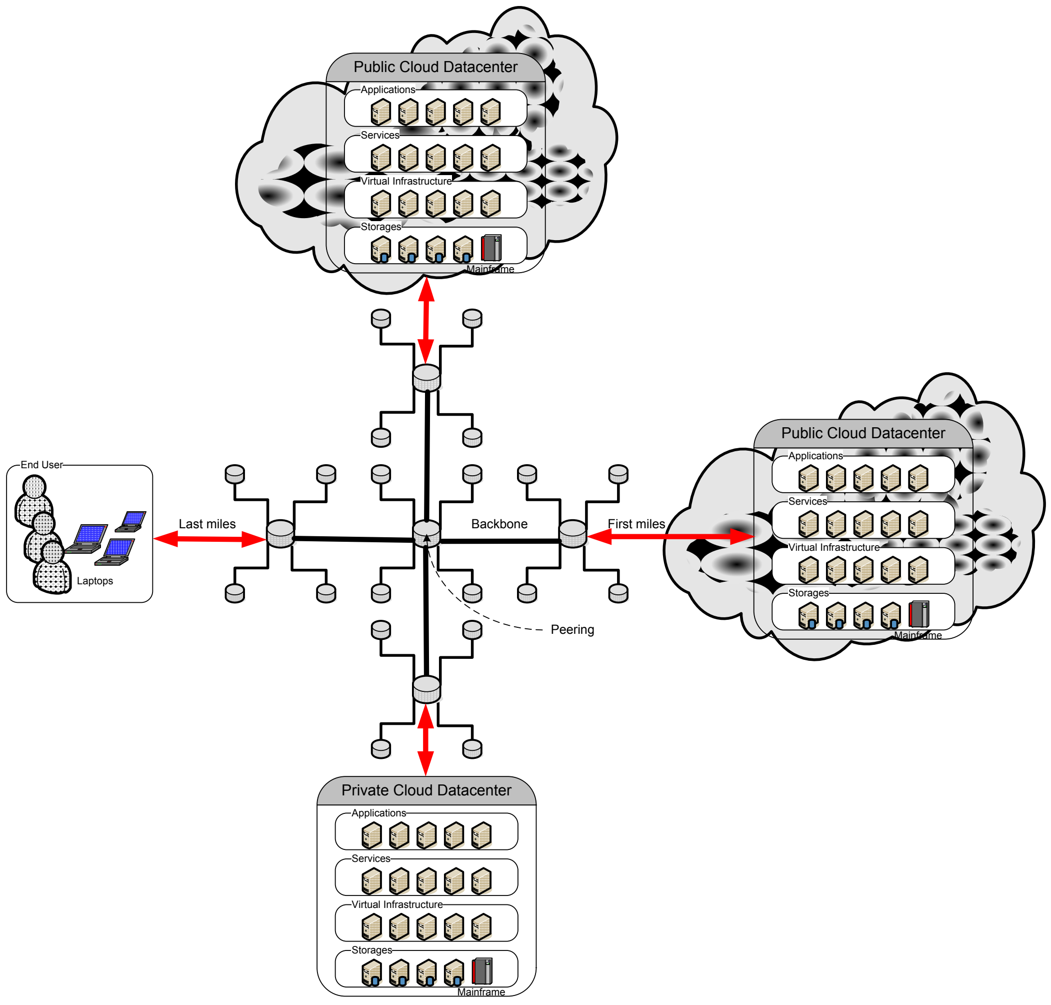


Figure 5.8. Cloud Performance Challenge

5.4.2 Cloud Scalability (CS)

The CS is another remarkable difference from traditional ESOA scalability. The tier-based ESOA architecture with traditional middleware does not scale linearly and does not allow applications to scale on-demand. In terms of framework, we define

$CS \equiv \exists\, hasMetric.(ScaleUp \cup ScaleOut \cup ScaleIntoCloud) \cap$

$\exists\, hasImpactFactor.(CloudServiceType \cup VMType \cup Cost)$,

in which

ScaleUp.metrics=performance (Server, Resources, Demand),

where ScaleUp = Scale Up or Vertical Scale is a traditional way to scale the system. The performance function increases for a given Server when Demand is increasing and Resources (CPU, Memory) are added to the Server. When the performance function is linear, either Throughput is doubled or ResponseTime is cut to half when the resources (CPU, processes, disk) are doubled for the given server computer. It is an ideal scalability, but it is hard to achieve in practice.

ScaleOut.metrics= performance (activeServers, onDemandServers, Demand)

ScaleIntoCloud.metrics= performance (activeServers, onDemandServers, Demand)

where ScaleOut = Scale Out or Horizontal Scale is a way to scale the system in the same enterprise datacenter. ScaleIntoCloud = Scale into Cloud or Global Scale is a new way to scale out the enterprise system. In the performance function, activeServers is a set of servers in a cloud system and onDemandServers is a set of passive servers in the server pool or queue which are waiting to be added to the system. For ScaleOut, the service pool is inside the enterprise datacenter, but for ScaleIntoCloud, the service pool is in the cloud service provider's datacenter.

The performance function is an increasing function or a non-increasing as well as non-decreasing function when Demand is increasing and a sub-set of passive servers are activated and added into the system for the same services, which means that the scale out system can either reduce ResponseTime or increase Throughput. In other words, it can keep the desired performance when the requirements of concurrent users are increasing.

There are several factors that impact cloud scalability. The CloudServiceType impacts the way and level of cloud system scalability. For IaaS, such as Amazon EC2, its scalability is through on-demand instances, while for PaaS, such as Google AppEngine, its scalability is through its auto scalable runtime. VMType (Virtual Machine Type) greatly impacts not only the way of scalability, but also other quality attributes and cloud system design. Table 5.4 shows the impacts. Public cloud utility computing nature also impacts scalability through indicator cost scalability. For example, the Amazon Auto Scaling service uses CloudWatch as the monitoring mechanism for determining to scale up or down. The cost of using Elastic Load Balancer is from \$0.025 per hour (\$18.25 per month).

Comparing the traditional way with the dynamic way to implement scalability from the Configuration perspective, the scalability of the traditional style can be described as follows:

$TS = ESOA_{scale}$(resources, connections, deployment, configuration).

Here, all parameters are tied to a workload estimate and are not allowed to be changed dynamically on-demand. The cloud scalability *CS* greatly improves *TS*, which allows applications to scale on-demand – scale up and scale down resources and network connections dynamically. The *CS* can be described as

$CS = ECSA_{scale}$(resources (config), connections(config), deployment(config),

config(t, request, policy))

which means that the resources, such as OS, CPU, memory, and networking capacity and redeployment, depend on a dynamic reconfiguration operation (config) which depends on time *t*, on-demand request, and policy. The *CS* can be implemented by an SLA-driven IaaS type service [206]. The *CS* has elastic scalability, such as Amazon AWS (EC2, S3) and IBM PowerVM which are all based on the scale-out principle. In the following, we show that the scale-out is a better way to get cloud scalability.

If we assume that the cloud system is continuous and that some of its attributes are derivable, then dynamic scalability can be modeled by a mathematical elastic equation:

$$\mathrm{Performance(t)} =_{\alpha} \frac{Capacity(t)}{Demand(t)}, \qquad (5.19)$$

In (5.19), α is a positive constant. *Capacity(t) = f(number_of_instances, number_of_threads, t).* The desired system performance is invariant with time *t* when *Demand(t)* is increasing or decreasing with *t*. Let us suppose both *Capacity(t)* and *Demand(t)* are derivable, then taking the derivative of *t* on both sides of (5.19), we have:

$$\frac{Capacity'(t)}{Capacity(t)} = \frac{Demand'(t)}{Demand(t)}, \qquad (5.20)$$

which means that if *Capacity(t)* and *Demand(t)* satisfy (5.20), the *Performance(t)* is invariant with time t. To satisfy (5.20), changes in the ratio of *Capacity(t)* with *t* should be the same as that of *Demand(t)* with *t.* Therefore, this proves that ScaleOut is a better way for achieving cloud dynamic scalability.

Table 5.4. Impacts on Attributes of VMType

VMType	**Scalability**	**Impacts on other Attributes**	**Service Type and Examples**
Instruction set-based VM	Through lower level on-demand instance	• Less manageability by service provider • Less built-in functionality • More Flexibility • More Portability • More service consumer control	IaaS Amazon EC2 [9]
Managed runtime-based VM	Through bytecode-level on-demand platform	• Medium manageability • Medium Flexibility and Portability • Medium built-in functionality	PaaS Microsoft Azure Service Platform [142]
Framework-based VM	Through high-level on-demand platform	• Higher manageability by service provider • Less Flexibility and Portability • More built-in functionality	PaaS Google AppEngine [236]

5.4.3 Cloud Security (CSE)

Security is a common constraint for any enterprise architectural style and a common concern for any enterprise architecture. Enterprises have even more security concerns on adopting public cloud services, since they are located in the provider's datacenters and accessed from the Internet. The information of companies is stored in the system over which the cloud service provider has no control. Therefore, for the ECSA architecture, the CSE is a very important quality attribute for design consideration on public cloud services and cloud service-oriented systems, as well as a very important checkpoint for evaluating an ECSA architecture. Based on our framework, CSE is defined as:

$$\text{CSE} \equiv \exists\, \text{hasMetric}.(\text{Confidentiality} \cup \text{DataSecurity} \cup \text{Vulnerability} \cup \text{Privacy}) \cap$$

$$\exists\, \text{hasImpactFactor}.(\text{CloudServiceType} \cup \text{CloudControlModel} \cup$$

CloudSecurityManagement) $\sqcap$

$\exists$ hasQualityDependency.(Performance $\sqcup$ Cost)

in which

Confidentiality.metrics= $\frac{NSCG}{NSP} \times 00$,

where *NSCG* is Number of Security Check Gates which includes identity management and access control (such as role-based access control); *NSP* is number of security points in the system from end to end as shown in Figure 5.9.

DataSecurity.metrics=securityLevel(dataClass)

$$= \begin{cases} FLDS, & \text{if data requires highest security} \\ SLDS, & \text{if data requires medium security} \\ TLDS, & \text{if data requires lowest security} \end{cases}$$

in which *FLDS* is First Level Data Security, such as data encryption for data (such as credit card number) requiring the highest security; *SLDS* is Second Level Data Security, such as data validation and encoding for data (such as application data) requiring medium security; *TLDS* is Third Level Data Security, such as data validation for data (e.g., application data) requiring the lowest security.

[50] has defined several vulnerability metrics. We choose one of them – Vulnerability Scan Coverage (*VSC*) in this chapter:

Vulnerability.metrics=*VSC*

$$= \frac{Count(Scanned\_Systems)}{Count(All\_Systems\_Within\_Cloud\,\mathrm{Pr}ovider)}$$

Cloud security is also impacted by several factors:

- CloudServiceType – different types of cloud services have different architectural topology structures and different security check points.
- CloudControlModel – from Figure 5.6, cloud computing splits the control domain into three domains, namely, (1) *SC* domain, (2) a domain shared by both *SC* and *SPR,* and (3) *SPR* domain. Therefore, different types of cloud services have different system control models. All the components in a private cloud are controlled by the organization, so it has maximum security control. However, in a public cloud, service consumers have less control on their data and operations, so it has less security control and the security is highly dependent on the *SPR*'s security provision.
- CloudSecurityManagement – security management is one of the elements in SM^{II}. Therefore, providers must have security management as part of their ECSA architecture. Better security management can help achieve better Security. Moreover, Security quality attributes also influence the architectural design of security management. The security management should provide easy, virtual controls to manage firewall and security settings for applications and runtime environments in the cloud.

In architectural design, the quality dependency of cloud security should be taken into consideration. Performance is one of the quality attributes. To guarantee data security across clouds, some data is needed by using *FLDS*. However, data encryption reduces router and server performance. Once data is sent to *SC* side, the data decryption also reduces client side computing performance. Moreover, improper security architecture design reduces system performance. Cost is another quality attribute. To meet both security and performance requirements, *SC* has to pay a higher price. The tradeoff between security, performance, and cost is an important consideration of cloud architectural analysis and design.

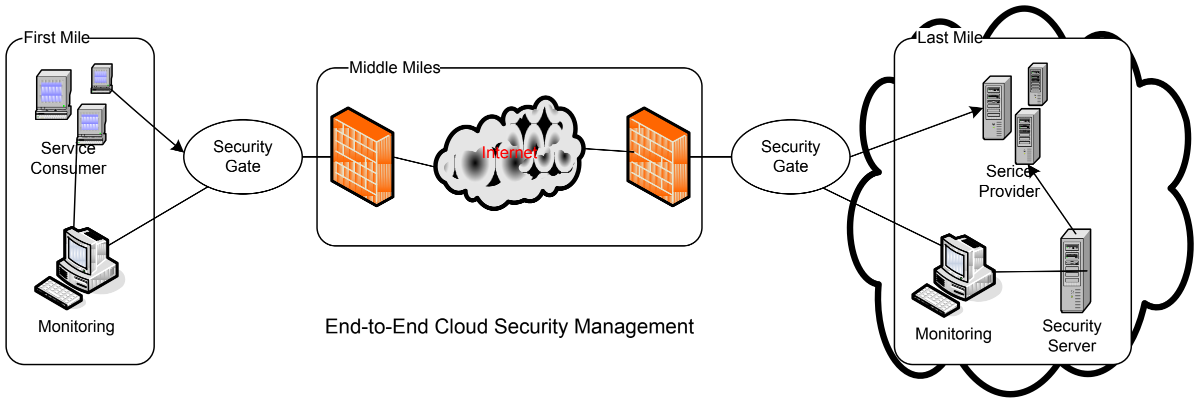


Figure 5.9. End-to-End Cloud Security Management

5.4.4 Cloud Service Availability (CSA)

The downtime of cloud service and system directly impacts enterprise business availability. The cloud provider needs to effectively receive and route incoming requests to the appropriate virtualized application instance on behalf of its customers. Google and Microsoft replicate each application instance to multiple physical locations. AT&T Synaptic Hosting spans multiple locations for its enterprise customers. Therefore, availability is a very important quality attribute. The term “high availability” is defined by the Institute of Electrical and Electronics Engineers (IEEE) as: “Availability of resources in a computer system, in the wake of component failures in the system.” A system can be called highly available if its applications and services are available even in the case of an error without direct human interaction. The Harvard Research Group (HRG) [85] classifies the availability as its Availability Environment Classification (AEC) as shown in Table 5.5:

In Table 5.5, the term "Highly Reliable" implies some degree of "Availability". If a system is not available, then the "Highly Reliable" is not possible. High availability is defined in enterprise architecture (EA) and traditional ESOA system by SLA [206]. It is also defined in SLA of public cloud services, such as the availability of Amazon EC2 is defined as 99.95% in its SLA [9]. Amazon's storage service S3 achieves fault-tolerance level availability. The framework proposed in Chapter 3 provides a formal way to analyze the CSA which can be defined as

Table 5.5. AEC and NINES (Number 9s)

AEC	*** Availability in %**
Disaster Recovery (AEC5)	99.99999%
Fault Tolerance (AEC4)	99.9999%
Fault Resilient (AEC3)	99.999%
High Availability (AEC2)	99.99%
Highly Reliable (AEC1)	99.9%
Conventional (AEC0)	99%
* The mapping to NINES is conducted by this paper.	

$$\begin{aligned} \text{CSA} \equiv\ & \exists\, \text{hasMetric.(SUT)} \cap \\ & \exists\, \text{hasImpactFactor.(CloudServiceType} \cup \text{CloudAvailabilityManagement)} \cap \\ & \exists\, \text{hasQualityRelation.(Reliability} \cup \text{Scalability} \cup \text{Consistency} \cup \text{ Performance} \cup \text{ Cost)} \end{aligned}$$

in which

SUT=Service Up Time=100% - SUA%,

where SUA is service unavailable rate:

$$\text{SUA} = \frac{\sum_{i=1}^{N} (OutageTime_i \times SDF_i)}{TotalActivityTime} \times 100 ,$$

in which SDF_i is Service Degradation Factor for the ith Outage and $0 \leq SDF_i \leq 1$, which indicates the service unavailable degree (such as partial service outage). The N can be a number of outages in a month or year.

Traditional Availability is defined based on server up and down times or defined through Mean Time Between Failures and Mean Time to Repair. In cloud service computing, a single server failure and repair should not impact the service availability. The CSA can be gained from cloud dynamic resource and availability management, such as Amazon EC2's availability zones which are within the same region and ensures complete and total redundancy for one's application [9]. Cloud service availability is impacted by several cloud service architectural style parts.

- CloudServiceType – Private Cloud Service availability can be achieved by its private IaaS. A hybrid cloud service availability can be increased by both private IaaS and public IaaS. Both PaaS and SaaS service availability can be met by using both *SPR*'s private IaaS and other public IaaS, such as Amazon's EC2. Therefore, IaaS type cloud service is a kind of service for other service availability.
- CloudAvailabilityManagement – CSA requires the service availability (resource) management (CAM) which is one of the elements in SM^{II} in ECSA style and better CAM can increase cloud service availability.

CSA as a quality attribute has relationship with other quality attributes. The relationship between CSA and cloud service reliability (CSR) can be defined as

$$CSA = \frac{MTTF}{MTTF + MTTR} = \frac{\mu}{\lambda + \mu},$$

where *CSR(t)* = Cloud Service Reliability over a time period *t* and *MTTF=1/λ* is "mean time to failure" and represents the average time until the first failure occurs, and *MTTR=1/μ* is "mean time to repair" and represents the average time required for repair, including any time to detect that there is failure, to repair the failure, and place the system back into operational state. *CSR(t)* = $e^{-\lambda t}$, so CSR is a decreasing function of λ. It is easy to show that *CSA* is a decreasing function of λ and an increasing function of μ. Therefore, it means less reliability, then less availability. Consider the triple quality attributes = (Consistency, Availability, Scalability). According to the CAP principle [130], for a shared data system (most of web application systems and cloud systems), the strong consistency (ACID-based consistency) could not be reached if the system wants to meet both high availability and the ability to tolerate network partitions. To achieve high cloud scalability, the system needs to be partitioned. Hence, in architecture design, one of the two attributes, Availability or Consistency can be chosen. Therefore, the ECSA architecture should have a better tradeoff between them. For the checkout process, one always wants to honor requests to add items to a shopping cart because it is revenue producing. In this case, one can choose high availability. Errors are hidden from the customer and sorted out later. However, when a customer submits an order, one favors consistency because several services--credit card processing, shipping and handling, reporting--are simultaneously accessing the data.

Availability is related to Performance. Most technologies, such as cluster and resource redundancy, are helpful at increasing performance. However, the fail-over and replication for high availability may sometimes increase temporal latency. Moreover, to reach high availability in the cloud, there are two kinds of cost to service providers: (1) cost of redundancy architecture should be taken into consideration; (2) penalty due to unavailable services, which is a motivation for *SPR* to consider higher availability.

5.4.5 Cloud Service Reliability (CSR)

While CSA specifies the cloud service uptime, CSR is defined as an ability to deal with external failures, while the cloud service continues to perform its functions in the runtime environment. As one of the cloud service quality attributes, CSR can be formally defined as

$$\begin{aligned}\text{CSR} \equiv\ & \exists\,\text{hasMetric.(MTBF} \bigcup \text{FailureRate)} \bigcap \\ & \exists\,\text{hasImpactFactor.(CloudReliabilityManagement)} \bigcap \\ & \exists\,\text{hasQualityRelation.(Availability} \bigcup \text{Security} \bigcup \text{Cost)},\end{aligned}$$

where MTBF = Mean Time Between Failures is a traditional system (hardware and software) reliability metric. It is used as a metric of cloud service. A cloud service system failure reflects a cloud service level outage to service consumers that can only be restored through repair or redundancy. Another typical metric is

$$\text{FailureRate} = {}^{1}\!/_{MTBF}\,.$$

Obviously, CSR requires good cloud reliability management which is part of SM^{II} in ECSA style. As shown in Section 5.4.4, Availability is a function of Reliability, but it is possible for poor reliability to reach high availability. Moreover, less reliability in cloud service system increases the risk of security violations. FailureRate is one of the key service level agreements. If FailureRate > the rate defined in SLA, then cloud service consumer should get service credit from their billing cycle. For example, Amazon S3 defines the FailureRate as “Error Rate” which means: (i) the total number of internal server errors returned by Amazon S3 as error status “InternalError” or “ServiceUnavailable” divided by (ii) the total number of requests during that five minute period. The Cost to service provider, such as Amazon, is to give penalty money back to consumers.

5.4.6 Cloud Resiliency (CR)

One of the concerns for adopting public cloud service is the potential loss of customer data and information when failures or disasters occur. Another concern is how cloud computing can assure guaranteed performance levels under high stress load situations. Cloud resiliency is a new quality attribute to address these concerns. The notion of resiliency has been studied in dependable computing but mainly with regard to fault-tolerance. Cloud computing systems operate in a highly distributed and dynamic environment. The changes are everywhere and at all times. The changes come not only from single-point failures and security attacks, but also due to user demands. Therefore, cloud computing requires not only fault-tolerance and security attack-tolerance, but also high-stress load-tolerance from unexpected operational demands. This constitutes cloud resiliency and is a very important requirement for cloud computing, especially for public cloud services to guarantee business continuity in the cloud. Therefore, Resiliency is one of the key non-functional constraints in ECSA style. This can be described based on our ontology framework:

$$\begin{aligned}\text{CR} \equiv\ & \exists\,\text{hasMetric.(Fault-Tolerance} \cup \text{SecurityAttack-Tolerance} \cup \\ & \quad \text{HighStressDemand-Tolerance)} \cap \\ & \exists\,\text{hasImpactFactor.(CloudServiceType} \cup \text{CloudResilientManagement)} \cap \\ & \exists\,\text{hasQualityRelation.(Reliability} \cup \text{Availability} \cup \text{Security} \cup \text{Scalability)}\end{aligned}$$

Fault-Tolerance.metrics1=Number of single point failure

Fault-Tolerance.metrics1=Duration of Failover

Fault-Tolerance.metrics2=Duration of Disaster Recovery (DR)

Fault-Tolerance.metrics3=TimeWinow of Fault Outage

SecurityAttack-Tolerance.metrics=Duration of recovering from attack

HighStressDemand-Tolerance.metrics= How quickly can the system allocate resources and maintain the desired system performance

CloudServiceType is a factor that impacts resiliency since different types of cloud services have different control boundaries as shown in Figure 5.6. The CloudResilientManagement is another factor that impacts resiliency. Better change, resource, and security management are all helpful in improving CR.

Moreover, this attribute has relationships with other attributes. Improvement in any of the Reliability, Availability, Security, and Scalability attributes can increase resiliency. Conversely, resiliency can improve system Reliability, Availability, Security, and Scalability, since it improves failover across a service. Traditional failover is to failover and perform disaster recovery (DR) from one server to another server at the same location (same data center) or at different datacenters in the same enterprise. Cloud failover or DR extends this to failover or DR to cloud. The ECSA style allows failover or DR to be in different datacenters at different geographic locations. For instance, Amazon EC2 is currently available in four regions or datacenters: US East (Northern Virginia), US West (Northern California), EU (Ireland), and Asia Pacific (Singapore). Each region (datacenter) consists of multiple Availability Zones (AZs). Amazon EC2 is able to place instances in multiple locations - multiple regions and multiple AZs. AZs are distinct locations that are designed to be insulated from failures in other AZs and provide inexpensive, low latency network connectivity to other AZs in the same Region. By launching instances in separate AZs, the EC2 consumers' applications can be failed-over to different AZs if their applications experience failures in one location. Thus, Amazon's resiliency keeps applications from being impacted by the failure of a single location. However, Amazon's approach missed a scenario – all AZs could be failed in a region or datacenter. In this case a single point failure can occur. We discuss this scenario in Section 7.3.4.

In this chapter, we have focused on specifying and analyzing the most important cloud quality ontology of ECSA. Some of the other quality attributes, such as Cloud Accountability (CAC), Cloud Flexibility (CF), Cloud Visibility (CV), Cloud Agility (CA), Cloud Interoperability (CI), Cloud Portability (CP) and Cloud Compatibility (CCP) are not discussed here, but we include them as part of the ECSA quality ontology. In summary, the cloud specific quality attributes can be described as

SQ^{II} = {CSP, CS, CSE, CSA, CR, CAC, CF, CV, CA, CI, CP, CCP}.

5.4.7 Public Cloud Service Properties

We define the properties of traditional web services in [201][204]. In this section, we specify the following major enterprise cloud service properties.

- Abstraction – This is shown in Table 5.6.

Table 5.6. Cloud Service Abstraction

Cloud service type	**Abstraction**
SaaS-type services	Abstraction of any computation and application in the cloud
PaaS-type Services	Abstraction of underlying hardware, software, tools and application development environment
IaaS-type Services	Abstraction of underlying hardware resources and infrastructure. It decouples the workloads and payloads of other enterprise datacenters from the physical infrastructure and manages the abstraction instead of the infrastructure.

- Standard service interfaces or contracts

Cloud computing delivers services to consumers through standard ESOA style service interfaces, such as web service API and SOAP messaging, RESTful service [202] API and HTTP protocol, event-driven service interface, or other well-defined service interfaces. Figure 5.10. Service Interfaces of Amazon S3Figure 5.10 depicts the interfaces of Amazon storage service S3.

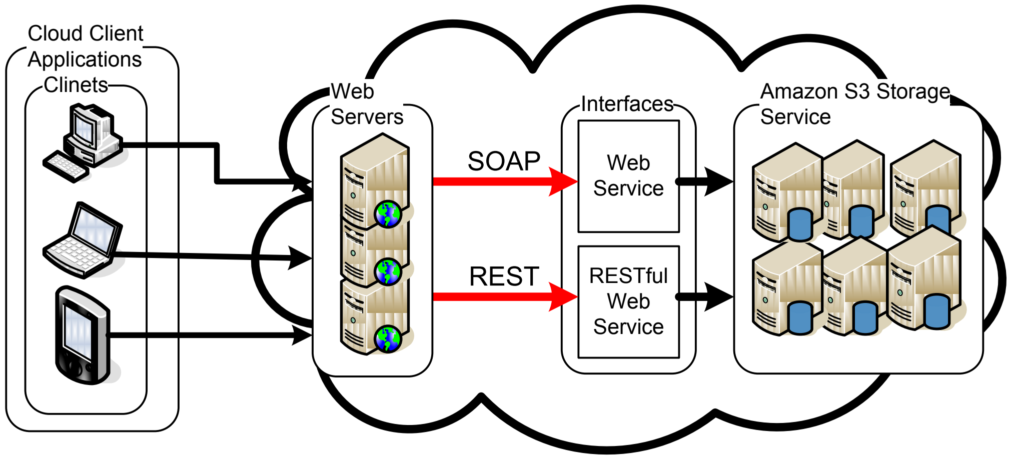


Figure 5.10. Service Interfaces of Amazon S3

- Loose Coupling

Loose coupling is an important property of traditional ESOA web services. It is also a very important property of cloud services. This property is one of the service design principles [67]. It is also a criterion for evaluating an ECSA architecture. The reason why it is important is that tight coupling results in expensive cloud service agility, reliability, and scalability for enterprise systems. The loose coupling principles and technologies of ESOA style service, such as asynchronous messaging and messaging queues, can help cloud services to achieve the property.

- Autonomy

Autonomy represents the ability to self-govern, which is one of the principles of SOA service design [67]. The principle is also one of the properties of enterprise cloud services. The on-demand self-service is one of the key characteristics of cloud computing, which means that cloud services should be able to allow consumers to unilaterally provision computing capabilities, such as server time and network storage, as needed without requiring human interaction with each service provider. If a cloud service lacks self-government, it is hard for it to be adopted by other enterprises. For instance, in Amazon S3, a public storage cloud has been designed such that individual components can make decisions based on local information.

- Reusability

Traditional ESOA services are reused by their owner and owner's partners. Public cloud services can be reused by not only the service provider, but also by all service consumers who subscribe to the services.

- Statelessness

Statelessness is another important property of ESOA services [67]. It is even more important for enterprise cloud services due to the more dynamic nature of cloud computing. The statelessness does not mean that there is absolutely no state. Services should keep as little of the computing states or service activities as possible. "There are different levels of statelessness a service design can achieve, depending on the frequency of state deferral and the quantity of state data being deferred. These levels are usually specific to each service capability"[67]. Because the components in cloud services are becoming increasingly transient, they can not support persistent state data. Cloud service should be as stateless as possible by pushing the state data out of the service and separating processing and data as much as possible. For example, Amazon Alexa cloud web search service uses SimpleDB to store process states [226].

- Composability

In Section 5.3.7, we have shown that individual cloud services can be composed, integrated, or aggregated into high-level service processes or workflows for completing a complex enterprise task in clouds. Cloud service composability is essentially the extension of one of the SOA principles into the cloud.

- Discoverability

Discoverability is another extension of ESOA service property to cloud services. The enterprise cloud services should be discoverable and interpretable for consumers. Many cloud providers extend SOA discoverability by extending the SOA registry technology, such as IBM WebSphere service registry and Microsoft Azure's cloud service registry. We propose the following ECSA dual triangles architecture shown in Figure 5.11 which shows a cloud registry for cloud services. In the dual triangles, the first triangle is a traditional SOA triangle and the second triangle is a cloud service triangle which consists of a cloud service consumer, a cloud gateway for connecting to public cloud services and a cloud service registry. The two triangles are connected through two registries and the service consumer. The cloud services can be discovered through the cloud registry.

- Subscription

The subscription is a property only for public cloud services. Traditional ESOA services and private cloud services serve internal consumers of the service provider as part of the ESOA system through internal service contracts without paying fees for services. However, the public cloud services are serving consumers of other enterprises. The cloud services are executed through a set of provisioning and subscription services [243].

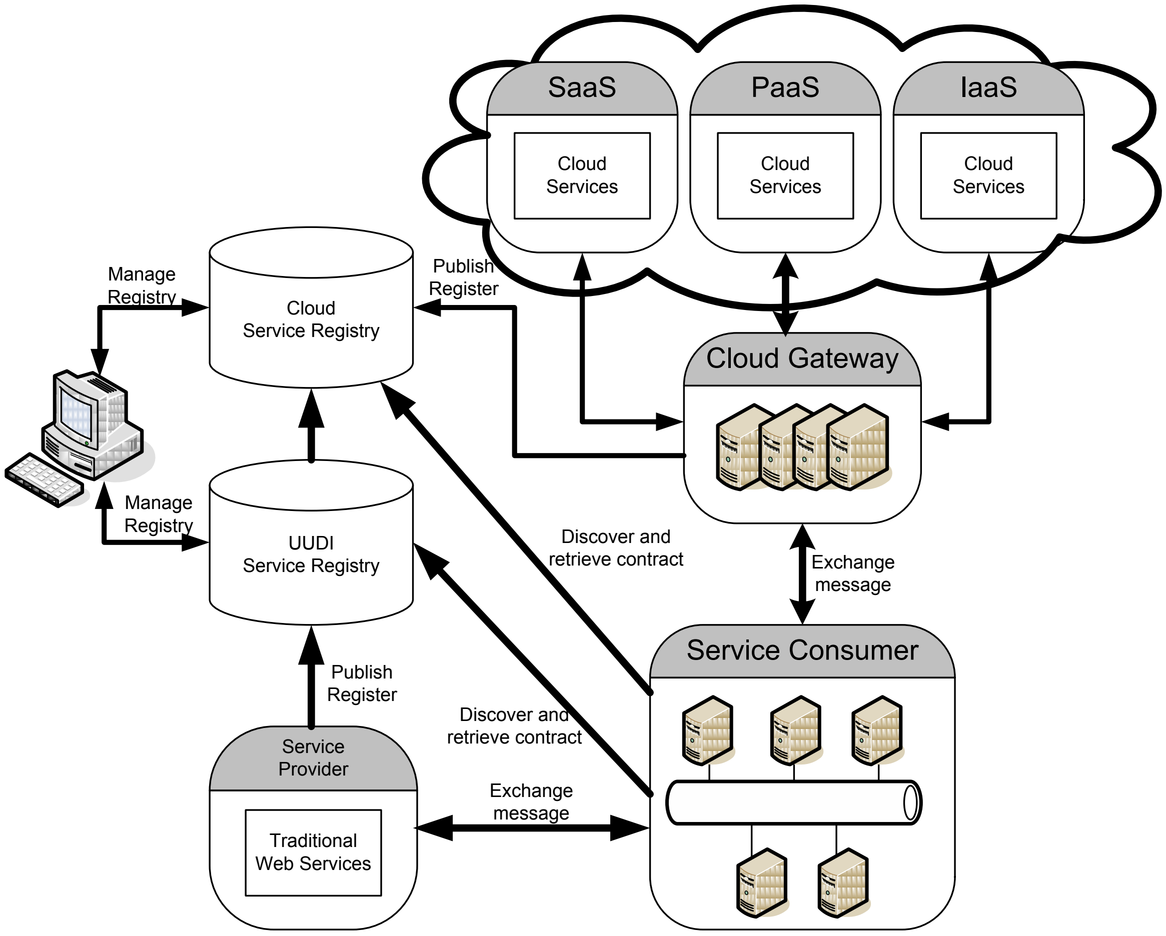


Figure 5.11. Dual Triangles

In this section, we defined cloud quality attributes and discussed several important quality attributes for cloud computing systems. We also described the major properties of public cloud services. The cloud quality attributes and service properties are non-functional constraints of ECSA architectural style, and provide the basis for designing and evaluating the ECSA architecture.

5.5 Summary

In Chapter 5, the ECSA style ontology is defined and specified. Specifically, the ECSA quality ontology is defined and described by using decscription logic and metrices. The ontology-based modeling and analysis are very helpful at understanding complicated enterprise cloud service architecture and guiding ECSA style system design and analysis.

CHAPTER 6

SLA-AWARE ENTERPRISE SERVICE COMPUTING

There is a growing trend towards enterprise system integration across organizational and enterprise boundaries on the global Internet platform. The Enterprise Service Computing (ESC), such as ESOA and ECSA we defined in Chapters 4 and 5, has been adopted by more and more corporations to meet the growing demand from businesses and the global economy. However, the ESC as a new distributed computing paradigm poses many challenges and issues of quality of services. For example, how is ESC compliant with the quality of service (QoS)? How do service providers guarantee services which meet service consumers' needs as well as wants? How do both service consumers and service providers agree with QoS at runtime? In this chapter, SLA-Aware enterprise service computing is first introduced as a solution to the challenges and issues of ESC. Then, SLA-Aware ESC is defined as new architectural styles which include SLA-Aware Enterprise Service-Oriented Architecture (ESOA-SLA) and SLA-Aware Enterprise Cloud Service Architecture (ECSA-SLA). In addition, the enterprise architectural styles are specified through our extended ESOA and ECSA models. The ECSA-SLA styles include SLA-Aware cloud services, SLA-Aware cloud service consumers, SLA-Aware cloud SOA infrastructure, SLA-Aware cloud SOA management, SLA-Aware cloud SOA process and SLA-Aware SOA quality attributes. The main advantages of viewing and defining SLA-Aware ESC as an architectural style are (1) abstracting the common structure, constraints and behaviors of a family of ESC systems, such as ECSA-SLA style systems and (2) defining general design principles for the family of enterprise architectures. The design principles of ECSA-SLA systems are proposed based on the model of ECSA-SLA. Finally, we discuss the challenges of SLA-Aware ESC and suggest that the autonomic service computing, automated service computing, adaptive service computing, real-time SOA, and event-driven architecture can help to address the challenges.

6.1 Introducing SLA-Aware Enterprise Service Computing

Enterprise Service Computing (ESC) is a new distributed computing and architectural style that has been adopted by more and more enterprises. ESC primarily includes Enterprise Service-Oriented Architecture (ESOA) [201][203][204] and Enterprise Cloud Service Architecture (ECSA) [205]. Because of complicated business requirements and high customer demands, ESC poses many challenges and issues, such as performance (latency, loss, and jitter) and dependability (security, trust). The Quality of Service (QoS) becomes crucial for ESC to achieve its vision and meet business requirements and customer demands.

Nowadays, most enterprises will only invest in IT when there is a clear return on investment, lower total cost of ownership, and a clear demonstration of cost savings. Investments made in services, web services and cloud service initiatives offer the opportunity to realize these requirements, but these investments need to be deployed in a consistent, repeatable, and manageable fashion. Traditional operation management is incapable of offering the unique management functionality that can help achieve these requirements as compared to service-oriented management which is based on QoS.

Service Level Management (SLM) is one of the most important and fundamental service-oriented management techniques. SLM provides mechanisms and tools for managing individual services and the SOA processes composed of a set of services designed to meet the QoS requirements and demands of enterprises and their customers. The Service Level Agreement (SLA) is a specification of service or service process functional provisioning and non functional goals - QoS which is agreed to by both service providers and service consumers. The Service Level Objectives (SLO) are key elements of SLA, which are specific and measurable quality attributes in the SLA, such as availability, throughput, frequency, performance (response time), and other quality attributes. SLA has been employed in industry such as networking and telecommunications for several decades. However, adoption of dynamic SLA in ESOA systems is relatively immature and suffers from lack of standards. Recently, cloud computing and ECSA have become the next generation enterprise service computing. The SLA and SLM have become more and more important because of the dynamic service computing environment and infrastructure. Dynamic and automated SLM provides an SLA-Aware approach in ESOA or ECSA architecture. An architectural style is a coordinating set of architectural constraints. The SOA quality attributes are the architectural constraints of ESOA and ECSA. The QoS and SLA can be part of architectural constraints and contracts at the service level in ESOA and ECSA. Therefore, at the architectural style level, adding SLA-Awareness to ESOA or ECSA generates a kind of specific architectural style, which is called SLA-Aware ESOA or SLA-Aware ECSA. At the ESOA and ESCA system (instance) level, the approach allows SLA to play a QoS role between each service consumer and service provider, which greatly improves the service visibility. It also brings service quality control intelligence and capacity into ESOA or ESCA systems, so that it greatly enhances SOA management capabilities. Therefore, ESC can meet service or service process functional provisioning and non-functional goals – QoS so that service providers satisfy service consumers with specific services. In addition, enterprises gain revenue from the services and avoid troubles caused by disputed services.

In this chapter, we first discuss the challenges and issues of ESC. Second, we discuss general QoS and SLA concepts, their ontology, standards (such as WS-Agreement), languages (such as WSLA), and classification in enterprise service computing. Third, we define SLA-Aware ESOA and ESCA architectural styles. The styles include:

- SLA-Aware SOA Quality Attributes

  The SLA-Aware quality attributes are fundamental to the design of SLO and SLA for ESC.

- SLA-Aware Services

  The measurable SLA quality attributes are the service constraints of which the service provider is aware in the service at runtime.

- SLA-Aware service consumers

  The service consumer is aware of the SLA and can visit it through client-side self-management portal.

- SLA-Aware service process

  The SLA-Aware SOA process consists of a set of SLA-Aware services for executing business processes. The SOA process itself is also aware of a process-wide SLA.

- SLA-Aware SOA infrastructure

  We define an SLA-Aware SOA infrastructure as a set of SLA-Aware infrastructure services such as SLA-Aware (or QoS-Aware) network services and SLA-Aware storage services.

- SLA-Aware SOA management

  SLA-Aware SOA management is defined as a set of SLA-Aware management services which provide SOA system services, including SLA management services, SLA monitoring/measuring services, SLA negotiation services, and SLA reporting services.

- SLA-Aware Cloud Service Provision and Subscription

  SLA-Aware cloud service provisioning and subscription will be discussed. The end-to-end SLA-Aware cloud service architectural style is also described

Finally, we discuss the extensions of ESOA-SLA and ECSA-SLA. In this chapter, we assume that all services are web services unless otherwise stated.

6.2 The Concept of SLA and SLA-Awareness

The existence of a quality service level agreement (frequently abbreviated as **SLA)** is of fundamental importance for any service delivery. It essentially defines the formal relationship between the service consumer and the service provider. We define SLA for SLA-Aware enterprise service computing as follows.

Definition 1: Service Level Agreement is a negotiable QoS contract between service consumer (SC) and service provider (SP) on the service guarantees for service consumers. The guarantees include the operations that need to be executed and the promised QoS that should be provided. Formally, we define SLA as

$$SLA = SLA\ (\mathrm{SC}, \mathrm{SP}, \mathrm{C(QoS)}), \tag{6.0}$$

in which SC is a service consumer or a service provided by another service provider, and C(QoS) is the negotiable QoS contract. Formula (6.0) can be simplified as $SLA = SLA$ (SC, SP), where the SP can be a web service or cloud service, such as IaaS [205]. There are two types of SLA according to its nature as shown in Table 6.1:

Table 6.1. Dynamic SLA vs. Static SLA

Type of SLA	Description	Machine processing	Measurement & Monitoring	Execution & Negotiation	Changing	Termination
Dynamic SLA	Defined by formal languages, such as WSLA, WS-Agreement	Yes	• Measure by SLA metrics and auto measure system • Monitoring by SLA monitor • Dynamic reporting	Dynamic SLM controls execution and negotiation between service provider and consumer automatically	Executing by dynamic SLM automatically	Executing by dynamic SLM automatically
Static SLA	Specified in a document	No	• Measure by SLA metrics • Monitoring by monitor	Traditional SLM is lack of automatic control and negotiation	Executing by traditional SLM manually	Executing by traditional SLM manually

Moreover, there are two types of dynamic SLA deployment as shown in Table 6.2:

Table 6.2. Vertical SLA vs. Horizontal SLA

Type of SLA	Definition from network layer prospective	Definition from enterprise architecture layer prospective
Vertical SLA	A SLA between two SPs or SC and SP on different OSI layers, such as a SLA between VoD and its ISP	A SLA between two SPs or SC and SP on different enterprise layers, such as a SLA between web application in web server layer and web services in application server layer.
Horizontal SLA	A SLA between two SPs or SC and SP on same OSI layer, such as a SLA between two IP domains.	A SLA between two SPs on the same enterprise architecture layer, such as a SLA between two web services in a workflow process.

Definition 2: SLA-Awareness is a capacity and design principle to guarantee QoS provided by services. It uses dynamic SLA binding in a service computing system environment to achieve its goal. The capacity and quality of an SLA-Aware service computing system is controlled by dynamic SLAs and managed by dynamic SLM.

6.3 SLA-Aware ESOA and SLA-Aware ECSA

Software architectural style is an abstraction of a family of systems as a pattern of structural organization. An architectural style is a coordinating set of architectural constraints that restrict the roles/features of architectural elements and the allowed relationships among those elements within any architecture that conforms to that style. Therefore, architectural style is a kind of roadmap and guidance for analyzing and designing concrete architectures. We previously proposed a model of enterprise service-oriented architecture (ESOA) [204]. In this chapter, we extend the ESOA style to the following SLA-Aware ESOA style:

$$ESOA\text{-}SLA = \langle S_{SLA}, C_{SLA}, D_{SLA}, S_{SLA}I, S_{SLA}M, S_{SLA}P, S_{SLA}Q \rangle, \tag{6.1}$$

in which

$S_{SLA} = \{s_i \mid s_i$ is a SLA-Aware we service$\}$, (6.2)

$C_{SLA} = \{c_i \mid c_i$ is a SLA-Aware service consumer$\}$, (6.3)

$D_{SLA} = \{d_i \mid d_i$ is a SLA-Aware SOA data element$\}$, (6.4)

$S_{SLA}I = \{r_i \mid r_i$ is a SLA-Aware SOA Infrastructure$\}$, (6.5)

$S_{SLA}M = \{m_i \mid m_i$ is a SLA-Aware SOA Management$\}$, (6.6)

$S_{SLA}P = \{p_i \mid p_i$ is a SLA-Aware SOA Process$\}$, (6.7)

$S_{SLA}Q = \{q_i \mid q_i$ is a SLA-Aware SOA quality attribute$\}$, (6.8)

Using the notation "$\triangleleft$" defined in Chapter 3 to indicate the style extension relationship, we have:

ESOA $\triangleleft$ ESOA-SLA.

The new constraint set SLA is added to its parent style ESOA, and the constraints apply consistently to the new elements, such as dynamic SLM and machine-processable SLA. The style extension is a part of the architectural style refinement [168]. We will explore the style, style refinement analysis, and evaluation in the next Chapter.

In Chapter 5, we presented a new enterprise service architectural style, called Enterprise Cloud Service Architecture (ECSA), which is a hybrid style of ESOA and cloud computing. Here we extend this style to the following SLA-Aware style:

$$ECSA\text{-}SLA = \langle S_{SLA}, C_{SLA}, D_{SLA}, S_{SLA}I, S_{SLA}M, S_{SLA}P, S_{SLA}Q, S_{SLA}D \rangle, \quad (6.9)$$

in which

S_{SLA} = {s_i | s_i is a SLA-Aware cloud service}, (6.10)

C_{SLA} = {c_i | c_i is a SLA-Aware cloud service consumer}, (6.11)

D_{SLA} = {d_i | d_i is a SLA-Aware SOA cloud data element}, (6.12)

$S_{SLA}I$ = {r_i | r_i is a SLA-Aware SOA cloud infrastructure}, (6.13)

$S_{SLA}M$ = {m_i | m_i is a SLA-Aware SOA cloud management}, (6.14)

$S_{SLA}P$ = {p_i | p_i is a SLA-Aware SOA cloud process}, (6.15)

$S_{SLA}Q$ = {q_i | q_i is a SLA-Aware SOA cloud quality attribute}, (6.16)

$S_{SLA}D = S_{SLA}D^{I} \cup S_{SLA}D^{II} \cup S_{SLA}D^{III}$, (6.17)

where

$S_{SLA}D^{I}$ = {d | d is a building element of development}, (6.18)

$S_{SLA}D^{II}$ = {d | d is a service deploy type}, (6.19)

$S_{SLA}D^{III}$ ={d | d is a SLA-Aware service delivery model}. (6.20)

Using our notation, we have ECSA $\triangleleft$ ECSA-SLA. Since the ESOA architecture can be regarded as a part of the ECSA architecture in the private cloud, we will focus on specifying the SLA-Aware ECSA in the rest of this section.

6.3.1 SLA-Aware SOA Quality Attributes

We have defined SOA quality attributes *SQ* as constraints of ESOA and ECSA. The SLA-Aware SOA quality attributes $S_{SLA}Q$ in (6.8) or (6.16) are subsets of *SQ,* and they are both important to the services and measurable. They can be measured by monitoring tools and calculated by service level management service. The core service level quality attributes are classified into several QoS classes shown in Figure 6.1. The quality attributes are fundamental to the design of KQI, KPI, SLO and SLA (Please see Section 2.4.4).

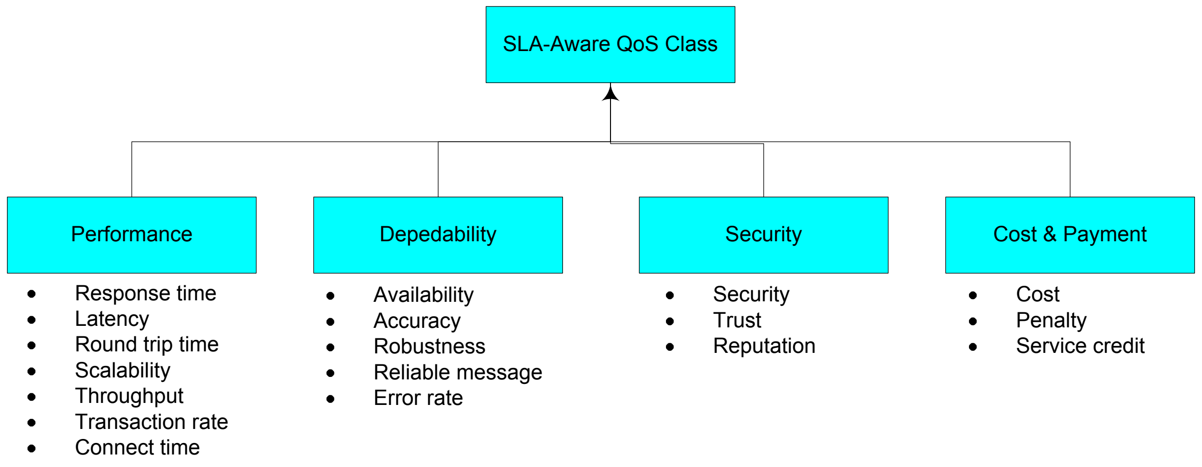


Figure 6.1. SLA-Aware QoS Taxonomy

6.3.2 SLA-Aware Web Services

Traditional web service is a self-contained software abstraction of business, technical functionality, or infrastructure management, characterized by a well-defined interface that focuses normally on the descriptions of functional aspects, such as input, output, preconditions and effects known as IOPE [62][63]. The interface of a web service is defined by the WSDL language. However, SLA-Aware web service not only focuses on its functional aspects, but also emphasizes its QoS through the dynamic SLA. We define SLA-Aware web service as follows:

Definition 3: The SLA-Aware Web Service in (6.2) or (6.10) is a web service described by both WSDL and formal SLA Language. It is managed by the SLM based on the dynamic SLA with its consumers. Figure 6.2 describes the SLA-Aware web service ontology:

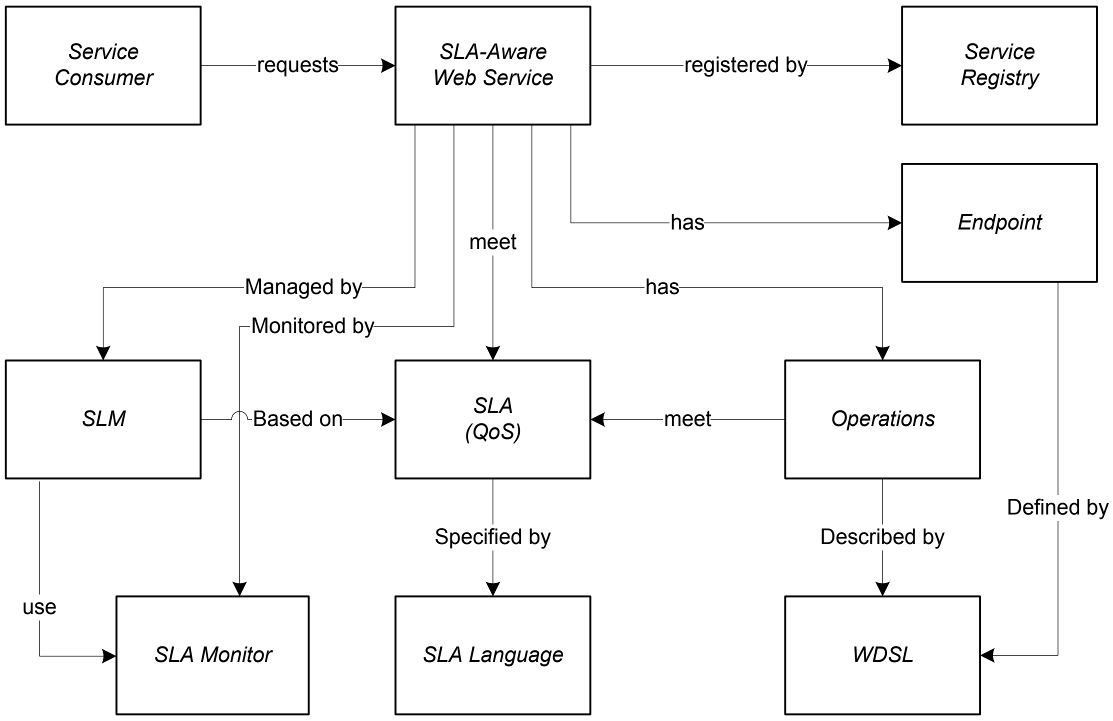


Figure 6.2. SLA-Aware Web Service Ontology

As discussed in Section 2.4.1, there are different languages, such as WSLA [129], WS-Agreement [128], which can be used for specifying SLA.

6.3.3 SLA-Aware Service Consumers

First, we need to extend the concept of service consumer, defined in (6.3) and (6.11) as follows:

$$\boldsymbol{C}_{SLA} = \boldsymbol{C}_{End} \cup \boldsymbol{C}_{S}, \qquad (6.21)$$

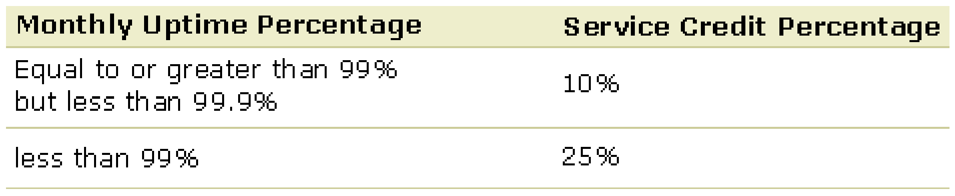
where C_{End} is a set of end service consumers in which the element can be any web service client or cloud web application, such as SaaS [205]; and C_s is a subset of S_{SLA}, in which the element is a service which consumes other services.

Unlike a traditional service consumer, an SLA-Aware service consumer is not only the service requestor, but also an SLA negotiator which sends an SLA negotiation request to the SLM of a service provider either directly or through a negotiation broker [87] before sending the service request. We define SLA-Aware service consumer as follows:

<u>Definition 4:</u> The SLA-Aware enterprise service consumer is a business application or another service which requests service from service provider(s), can initialize an SLA negotiation with its SP, and make decisions regarding service class and service request based on both functional and non-functional (QoS, such as performance, availability, security, pricing as well as penalty) requirements. The SLA-Aware service consumer should be self-managed through a self-service portal with a set of dashboards.

Figure 6.3 describes a model of the interaction between the SLA-Aware cloud service consumer (CSC) and the SLA-Aware cloud service provider (CSP). Moreover, we assume the CSP, such as Amazon S3 web service, is in the public cloud [205], which is based on a pay-as-you-go business model for its CSC. Therefore, there is service pricing in the service negotiation, billing service for handling CSC payment, and billing justification based on usage and agreement. For example, the SLA of Amazon web service S3 [8] defines the following Service Credit as its billing justification. Service Credits are calculated as a percentage of the total charges paid by you for Amazon S3 for the billing cycle in which the error occurred in accordance with the schedule as shown in Table 6.3:

Table 6.3. Service Credits of Amazon Web Service S3



Monthly Uptime Percentage	Service Credit Percentage
Equal to or greater than 99% but less than 99.9%	10%
less than 99%	25%

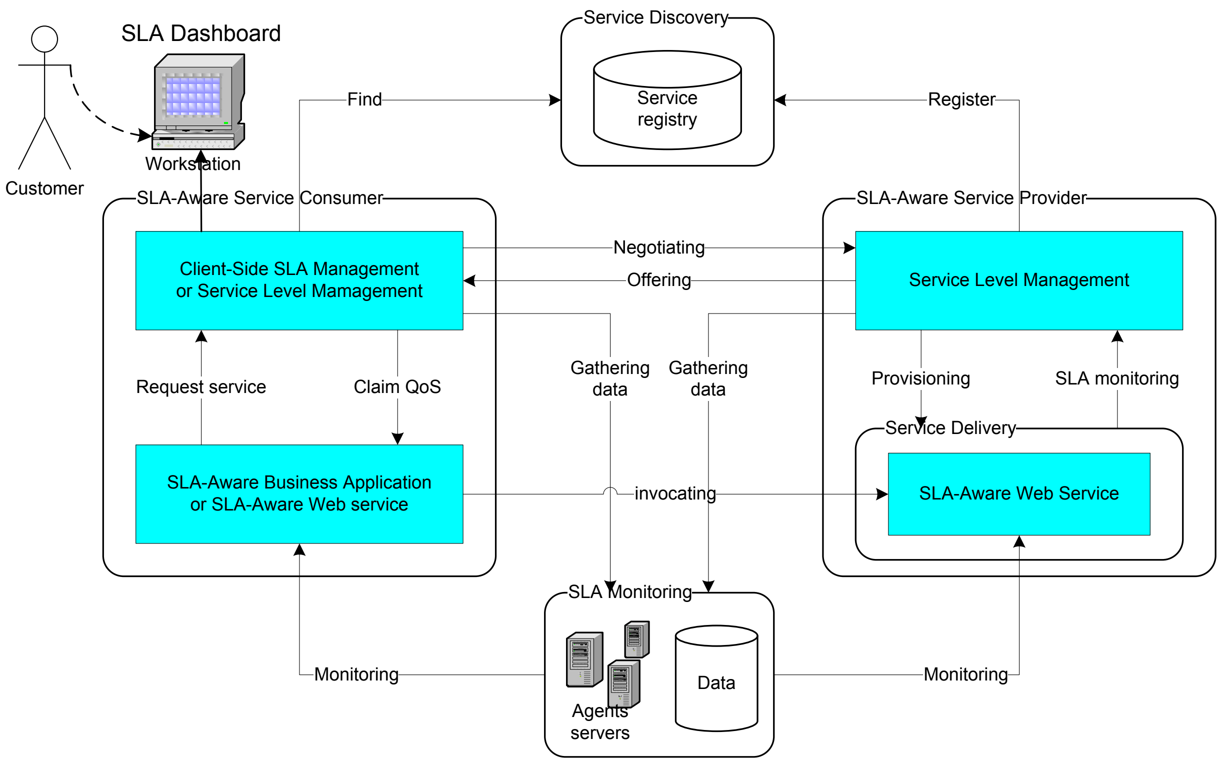


Figure 6.3. Model of the Interaction between SLA-Aware CSC and SLA-Aware CSP

In Figure 6.3, the Service Delivery can be middleware with web service containers, such as Weblogic and WebSphere application servers.

6.3.4 SLA-Aware SOA Infrastructure

The traditional SOA infrastructure is the heart of ESOA. It is the bridge of the transformation between business and services. However, the traditional enterprise SOA infrastructure is built in a kind of static data center (without adopting virtualization and other server consolidation technologies, like agility and alternate sourcing – cloud computing) in which (1) pre-provisioned resources are used - rigid, server silos and dedicated servers per application; (2) Server CPU utilization is often in single digits; (3) Scale is through adding hardware, and (4) Resources are shared within enterprise firewalls. Therefore, it is not adaptable to today’s on-demand business workload and real-time B2B requirements. It also costs more resources and power within an enterprise's data center. The SLA-Aware SOA cloud infrastructure is a kind of SLA driven service-oriented infrastructure which aims at improving the traditional SOA data center, reducing cost, and adapting on-demand requirements of business and customer.

Definition 5: The SLA-Aware SOA cloud infrastructure in (6.13) is an SLA driven service-oriented infrastructure with the following main characteristics:

- It is managed by SLA-Aware SOA management.
- It supports elasticity and dynamism – automatic scalability and load-balancing, failover based on SLA and in terms of virtualization [205] or other technologies [205].
- It supports global resource sharing through the Internet.
- It supports resource usage accountability – utility model [205].
- It can be a part of cloud service, such as PaaS type services (Google App Engine [205]), or can be a cloud service, such as IaaS type service (Amazon EC2).

Figure 6.4 is the high-level view of SLA-Aware Dynamic Data Center in a cloud service-oriented enterprise. The SLA-Aware SOA Cloud Infrastructure consists of

- An enterprise SOA and cloud service delivery network (SDN).
- A provisioning service.
- A dynamic virtualized infrastructure: Virtualization Infrastructure as a Service (VIaaS), such as VMWare.
- A physical resource infrastructure: Physical Infrastructure as a Service (PIaaS).
- Business Applications layer.
- SOA Infrastructure Management which includes SLA management as well as other management systems.

- Monitoring systems.

For an SLA-Aware SOA cloud infrastructure, the VIaaS and PIaaS should be able to manage resources, such as CPU, OS, networking and storage allocation and can tune and re-purpose resources to the environment. The SLA management should (1) guarantee the resources to be allocated dynamically based on demand; (2) guarantee QoS (such as availability, performance, and security) defined in SLA; and (3) guarantee the pricing and billing agreement. The Monitoring system should (1) monitor SLA and heartbeats; (2) monitor capacity of VIaaS and PIaaS; (3) monitor usage; (4) monitor utilization of resources as well as services; and (5) provide analysis and calculation results to SLA management and provisioning service as well as billing service.

In Figure 6.4, the other management aspects in SOA management may include service discovery, policy enforcement, etc. [204].

6.3.5 SLA-Aware SOA Management

The architectural styles ESOA-SLA and ESCA-SLA we have proposed are SLA-aware and service oriented. The SLA-Aware SOA management in (6.6) and (6.14) is one of the key parts in (6.1) and (6.9). It is different from the general concepts and approaches for SOA management of ESOA and ESCA that we have discussed, since it is SLA-Aware and dynamic. The ESOA-SLA and ESCA-SLA emphasize end-to-end SLA management.

Definition 6: The end-to-end SOA management ***SLAM*** can be defined as a set of SLM:

$$\boldsymbol{SLAM} = \{SLM_i \mid SLM_i \text{ is a SLM with } \mathrm{SLA}_i \text{ for service } s_i\}, \qquad (6.22)$$

in which s_i includes functional services, VIaaS, PIaaS, IaaS which are infrastructure services, and other SOA management services, such as security services and logging services. Figure 6.5 depicts the End-to-End SLA Management in service-oriented enterprise architecture.

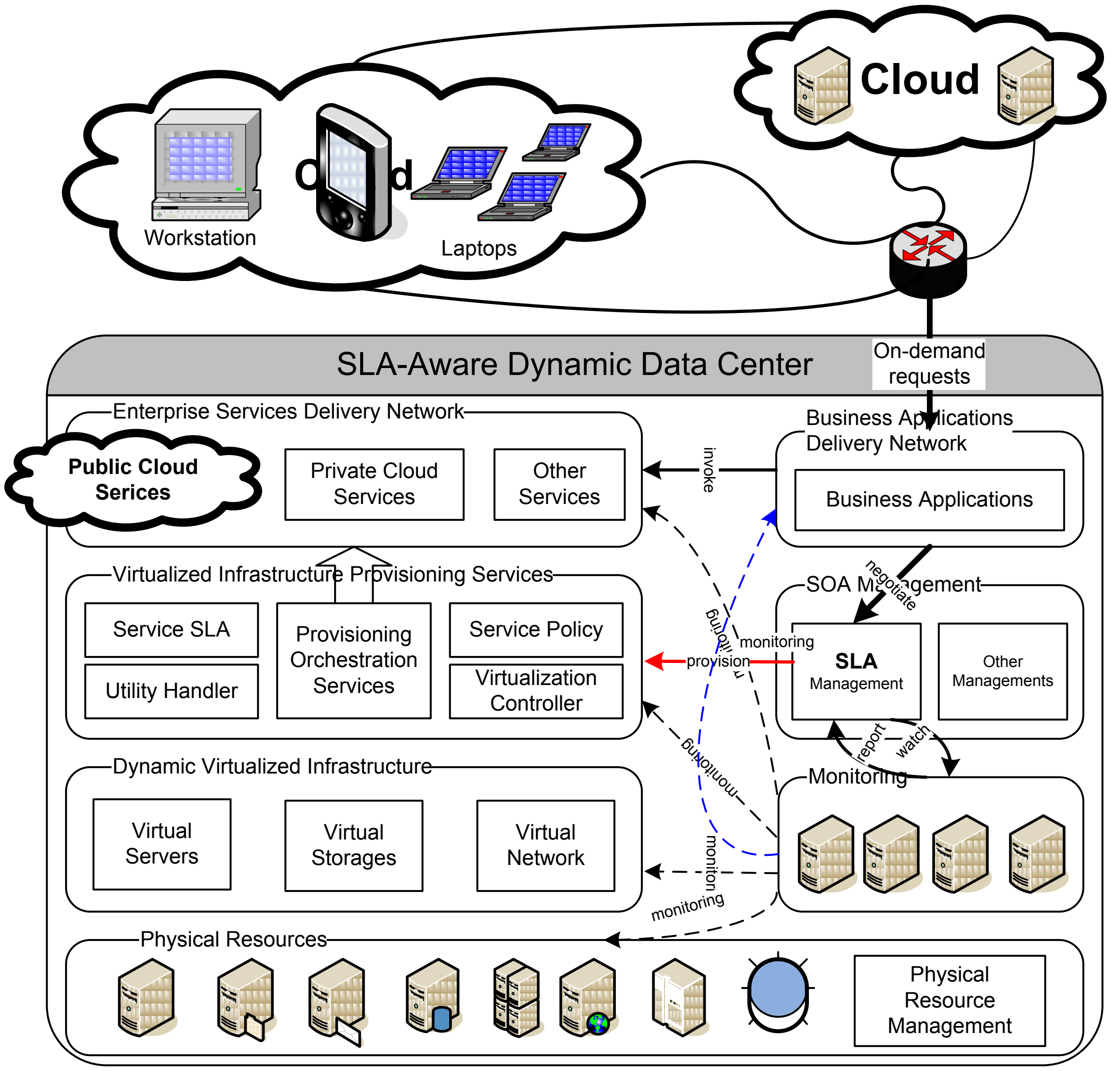


Figure 6.4. SLA-Aware SOA Cloud Infrastructure

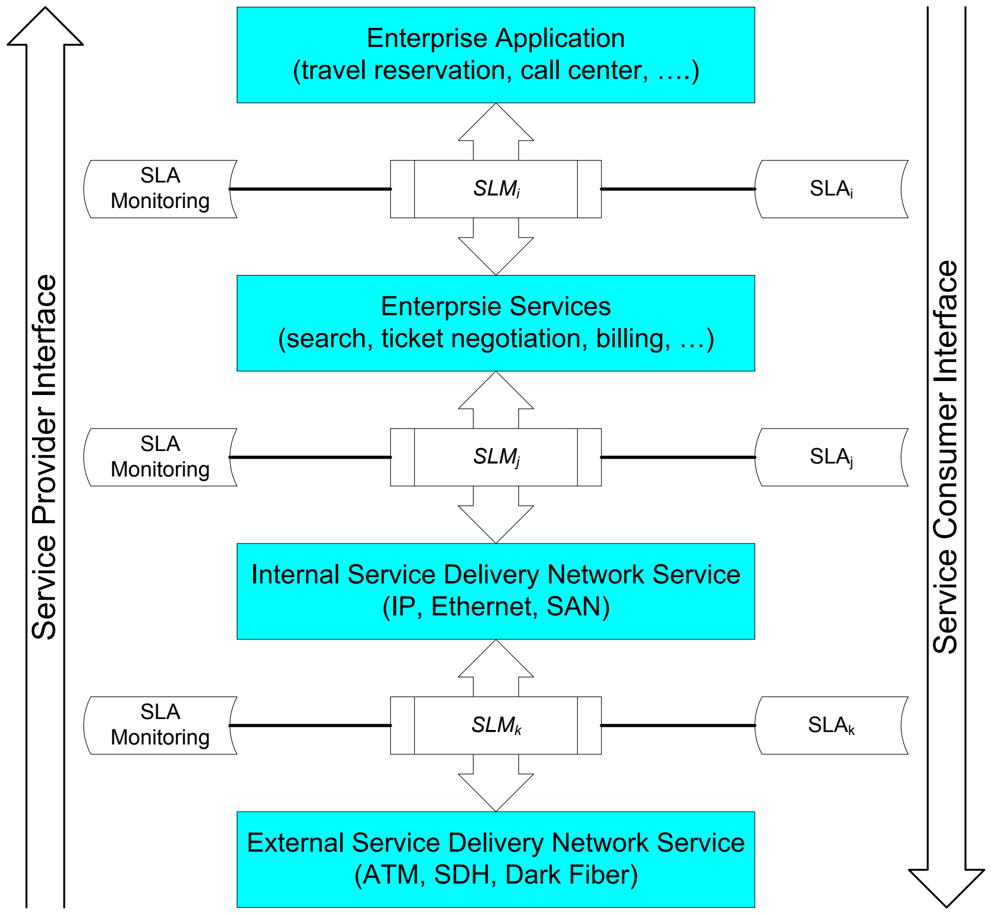
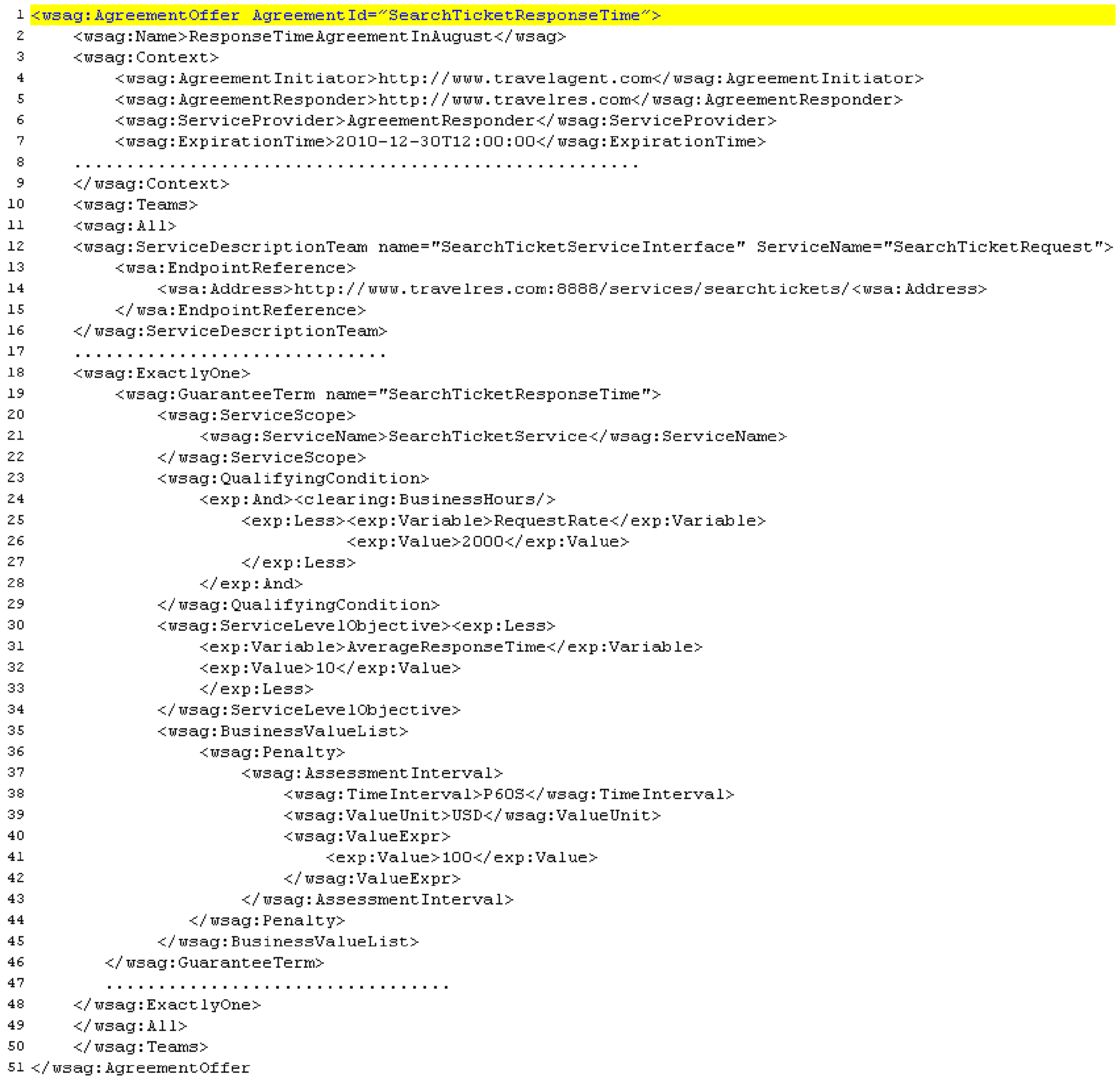

Service Provider Interface
Enterprise Application
(travel reservation, call center, ….)
SLA Monitoring
SLM_i
SLA_i
Enterprsie Services
(search, ticket negotiation, billing, …)
SLA Monitoring
SLM_j
SLA_j
Internal Service Delivery Network Service
(IP, Ethernet, SAN)
SLA Monitoring
SLM_k
SLA_k
External Service Delivery Network Service
(ATM, SDH, Dark Fiber)
Service Consumer Interface

Figure 6.5. End-to-End SLA Management in Service-Oriented Enterprise Architecture

Figure 6.5 shows that the SLM plays a service manager role. We highlight an SLA-Aware SLM for cloud service, such as the airline ticket reservation service in Figure 6.7. The SLM architecture can be implemented by WSLA framework [105], WSOL framework [212], or WS-Agreement standard [87]. For instance, the SLA negotiation and offer between service consumer and service provider can be implemented by WS-Agreement. Figure 6.6 is the agreement offer document defined by WS-Agreement for the ticket search service.



```
<wsag:AgreementOffer AgreementId="SearchTicketResponseTime">
    <wsag:Name>ResponseTimeAgreementInAugust</wsag>
    <wsag:Context>
        <wsag:AgreementInitiator>http://www.travelagent.com</wsag:AgreementInitiator>
        <wsag:AgreementResponder>http://www.travelres.com</wsag:AgreementResponder>
        <wsag:ServiceProvider>AgreementResponder</wsag:ServiceProvider>
        <wsag:ExpirationTime>2010-12-30T12:00:00</wsag:ExpirationTime>
    ......................................................
    </wsag:Context>
    <wsag:Teams>
    <wsag:All>
    <wsag:ServiceDescriptionTeam name="SearchTicketServiceInterface" ServiceName="SearchTicketRequest">
        <wsa:EndpointReference>
            <wsa:Address>http://www.travelres.com:8888/services/searchtickets/<wsa:Address>
        </wsa:EndpointReference>
    </wsag:ServiceDescriptionTeam>
    ..............................
    <wsag:ExactlyOne>
        <wsag:GuaranteeTerm name="SearchTicketResponseTime">
            <wsag:ServiceScope>
                <wsag:ServiceName>SearchTicketService</wsag:ServiceName>
            </wsag:ServiceScope>
            <wsag:QualifyingCondition>
                <exp:And><clearing:BusinessHours/>
                    <exp:Less><exp:Variable>RequestRate</exp:Variable>
                              <exp:Value>2000</exp:Value>
                    </exp:Less>
                </exp:And>
            </wsag:QualifyingCondition>
            <wsag:ServiceLevelObjective><exp:Less>
                <exp:Variable>AverageResponseTime</exp:Variable>
                <exp:Value>10</exp:Value>
                </exp:Less>
            </wsag:ServiceLevelObjective>
            <wsag:BusinessValueList>
                <wsag:Penalty>
                    <wsag:AssessmentInterval>
                        <wsag:TimeInterval>P60S</wsag:TimeInterval>
                        <wsag:ValueUnit>USD</wsag:ValueUnit>
                        <wsag:ValueExpr>
                            <exp:Value>100</exp:Value>
                        </wsag:ValueExpr>
                    </wsag:AssessmentInterval>
               </wsag:Penalty>
            </wsag:BusinessValueList>
       </wsag:GuaranteeTerm>
       .................................
    </wsag:ExactlyOne>
    </wsag:All>
    </wsag:Teams>
</wsag:AgreementOffer
```

Figure 6.6. Agreement Offer of WS-Agreement for Search Ticker Service

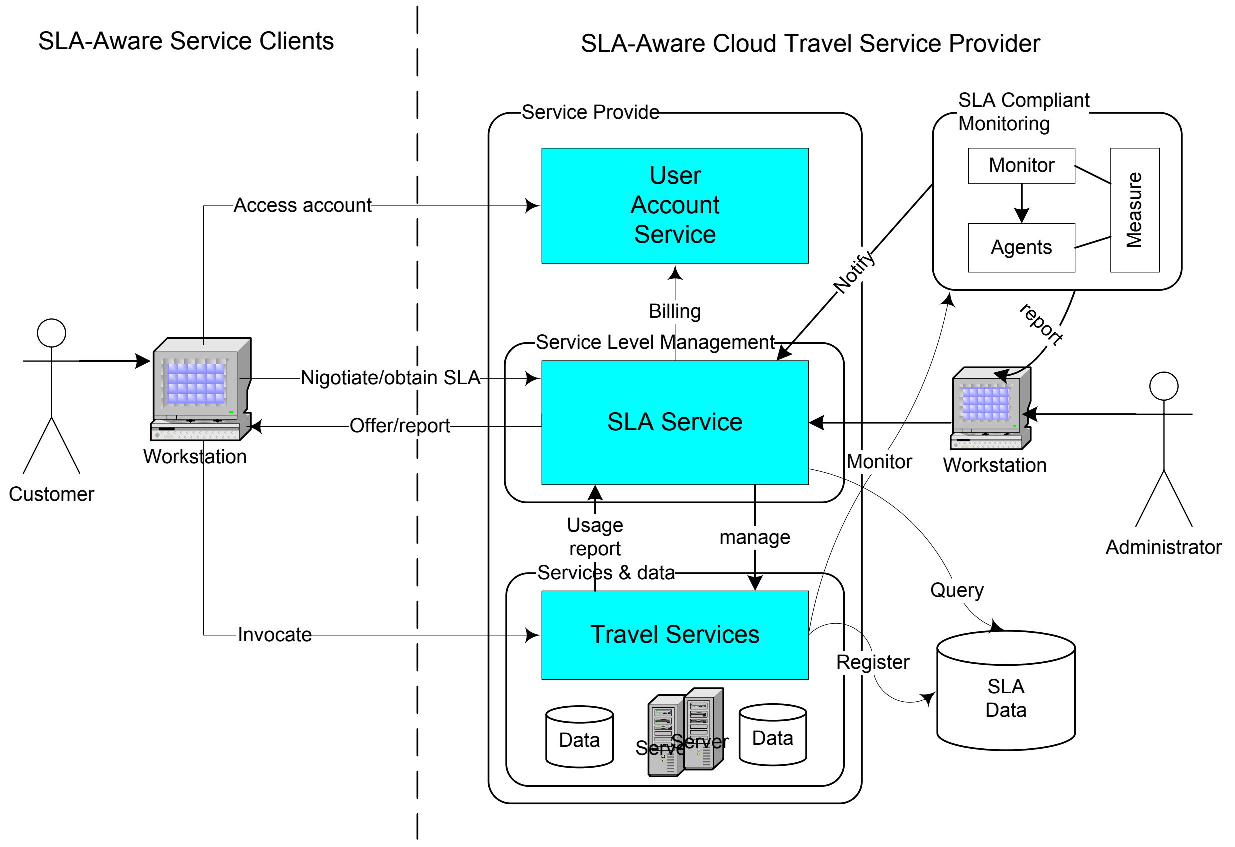


Figure 6.7. SLM for SLA-Aware Cloud Travel Service

A functional service like the travel service, under management of SLM, is different from traditional services. It must

- query the SLM when it is going to execute an action/operation (search tickets);
- notify the SLM of resource usage in a timely manner; and
- obey the SLM's instruction to destroy activities.

The user account service is one of the core parts for SLA management, since there is no way to handle users' credit and service payment without it. When the user proposes a new SLA, SLM needs to verify the user's credit from the account system. When the user uses the travel service, SLM needs to record the usage into the account service. At the end of each SLA billing cycle, SLM records the total usages in the user's account for billing the user. Moreover, when billing an account, if SLM finds that the account is suspended or closed, then the SLA will be suspended or closed.

6.3.6 SLA-Aware SOA Process

One of the important parts of the ESOA style is its set of SOA processes. The SOA process or workflow is an abstraction of Business Process Management (BPM). Each process is composed of multiple services in orchestration and/or choreography for completing a whole or partial business process or task. The traditional SOA process can be executed by using an ESOA infrastructure with a process engine in the internal network of an enterprise. However, traditional SOA processes face many challenges and issues: Real-time high performance (such as automated trading), on-demand scalability, large payloads (10+ MB), memory constraints, and high availability and reliability. The SOA process of ECSA style resolves the issues of traditional SOA processes. Some complex transaction processes and workflows in enterprises may need to compose multiple services in the cloud for completing the tasks. However, traditional ways lack end-to-end QoS guarantees for processes. The question is: How can the cloud process service provider guarantee the quality requirements from the service consumers. In this section, we specify the SLA-Aware SOA process $S_{SLA}P$ in ECSA-SLA.

Definition 7: Let $p_{SLA} \in S_{SLA}P$, then an end-to-end SLA-Aware SOA cloud process can be defined as

$$p_{SLA} = \{c \in C_{SLA}\} \cup \{ s_i \mid s_i \quad S_{SLA} \text{ and } i=1,2, \dots n\} \cup \{IaaS^k \mid IaaS^k \in S_{SLA}I \text{ , } k=1,2,\dots,m\}.$$

We define the end-to-end SLA chain for process p as

$$SLA_p = SLA(p) \cup SLA(IaaS) \text{ ,}$$

$$SLA(p) = \{ SLA_{ij} \mid SLA_{ij} = SLA(s_i, s_j), i \neq j, s_i \text{ is service consumer and } s_j \text{ is service provider, } SLA_{ij} \neq \emptyset\} \text{ ,}$$

in which $i=0,1,2,\ldots,n$, $s_0 = c$ is a service consumer which initiates the process. Suppose s_i is the first service called by c,

$$SLA(IaaS) = \{SLA_{i,IaaS}^{j} \mid SLA_{i,IaaS}^{j} = SLA(s_{i,\ IaaS}^{k}),\ i <= n,\ k = 1,2,\ldots,m,\ {}_{IaaS}^{k} \text{ is a service provider}\},$$

where $n \geq m \geq 1$, $SLA_{01} \neq \emptyset$, $SLA(p)$ can be empty and $SLA(IaaS) \neq \emptyset$, which means there are at least two SLAs – one is between the process service consumer and the process service, the other is between the process service and its infrastructure. If $n > 1$ and n is in the process p_{SLA}, s_i, $i=j_1,j_2,\ldots j_k$ are external services in the different clouds, then $m=k$. The structure of $SLA(p)$ depends on the process patterns and the way that the SLA_{ij} is specified. For instance, Figure 6.8 describes an SLA-Aware sequence travel reservation workflow with two cloud services. Therefore

$$p_{SLA} = \{c \in C_{SLA}\} \cup \{\, s_i \mid s_i \ \ S_{SLA} \text{ and } i=1,2,3,4\} \cup \{IaaS^k \mid IaaS^k \in S_{ct}\ I,\ k=1,2,3\},$$

$SLA(p) = \{\ SLA_{12}, SLA_{13}\}$, and

$SLA(IaaS) = \{SLA_{2,IaaS}^{\ 1}, SLA_{3,IaaS}^{\ 2}, SLA_{4,IaaS}^{\ 3}\}$.

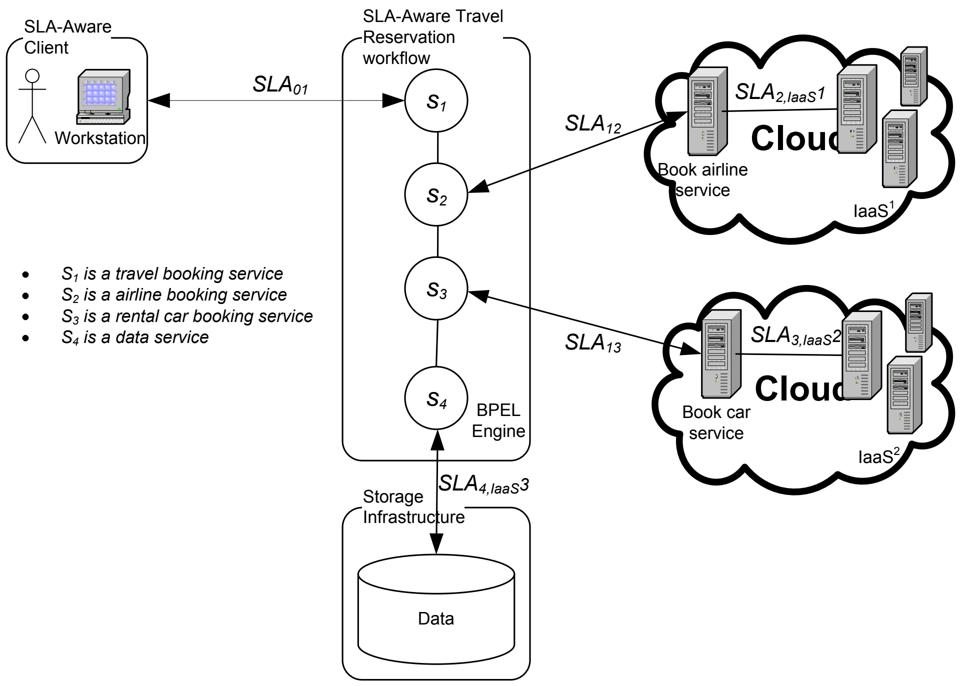


Figure 6.8. A SLA-Aware Sequence Travel Reservation Workflow

The SLA-Aware SOA Cloud processes, such as service composition, workflow, orchestration and choreography, are very important for improving customer experience and satisfaction with enterprises; therefore, this topic has attracted much research interest, including works listed in Section 6.1 and [242].

6.3.7 SLA-Aware Cloud Service Provisioning and Subscription

We previously defined the enterprise cloud services delivery model in Chpater 5. The extension of the SLA-Aware cloud service delivery model defined in (6.20) can be specified in the following Table 6.4:

Table 6.4. SLA-Aware Delivery Models of Cloud Services

Delivery Mode	**Description**	**Resource Sharing**
SaaS-SLA	SLA-Aware Software as a Service	Sharing software under dynamic SLA
PaaS-SLA	SLA-Aware Platform as a Service	Sharing platform under dynamic SLA
IaaS-SLA	SLA-Aware Infrastructure as a Service	Sharing infrastructure under dynamic SLA
IMaaS-SLA	SLA-Aware Information as a Service	Sharing information under dynamic SLA
IRaaS-SLA	SLA-Aware Integration as a Service	Sharing integration under dynamic SLA
XaaS-SLA	SLA-Aware other cloud service delivery models	Sharing other resources under dynamic SLA

All the SLA-Aware cloud services in different models are actually delivered through a set of SLA-Aware cloud service provisioning services [243] by service providers, which are part of the enterprise cloud SOA infrastructure. The SLA-Aware service provisioning in Figure 6.4 has four interfaces:

- An interface with SLA management (SLM), which accepts SLM control and reports service usage to SLM.
- An interface with resource management, which allocates resources for services based on the demand.
- An interface with service scheduling system to provide scheduled services to clients based on SLA.
- An interface with service consumers, which deliver services to consumers.

In an SLA-Aware SOA cloud service environment, the cloud service subscription [243] from clients (service consumers) is managed by a set of service provider's SLA-Aware service subscription services which process the subscriptions of service consumers with SLA information.

Zhang and Zhou pointed out that the cloud provisioning and subscription services should be extendable for supporting different types of resource sharing [243] and service subscription. The SLA-Aware service provisioning and subscription is a principle for designing ECSA-SLA style architecture as well as a challenge for both researchers and practitioners in enterprise service computing.

In summary, Section 6.3 primarily specifies the new architectural style (ECSA-SLA) and its ontology. The style emphasizes dependability within enterprise service computing through dynamic SLA mechanisms and SLA management as first-class architecture design considerations. However, we still face a lot of challenges in many aspects, especially in research and practice. We will discuss those challenges in the next section.

6.4 Challenges of SLA-Aware Enterprise Service Computing

SLA-Aware Enterprise Service Computing is a new enterprise architectural style. Higher automation, performance and adaptation are required for designing this style-architecture. Therefore, researchers and practitioners face a number of challenges. The challenges include:

- General Challenges:
  - Theoretical foundation of SLA-Aware enterprise service computing
  - Formalizing complicated service-oriented enterprise architectural styles
  - Verifying complex architectural styles
  - Autonomic self-service on the client-side, which can monitor and manage the SLA execution on the server-side
  - Automated service provisioning and subscription
  - Automated service discovery and selection
- New Challenges:
  - Automated service level management
  - Automated SLA monitoring which can monitor SLA execution dynamically
  - Adaptive resource management based on SLA and demand
  - Adaptive SLA-Aware service execution in SP environment, such as adaptive service performance and scalability management, change management, dynamic reconfiguration, exception management, and fault-tolerance
  - Adaptive system optimization

- Real-Time (RT) or close to RT SLA management, dynamic SOA Infrastructure and management.

Autonomic computing, automated and adaptive service computing and event-driven and real time service computing have been researched and adopted for tackling some of the challenges of SLA-Aware ESC. We have discussed the research work in Section 2.4.5 and Section 2.4.7.

6.5 Summary

In this chapter, we have introduced the SLA-Aware enterprise service computing and specified two new architectural styles: SAL-Aware ESOA and SLA-Aware ECSA in SLA-Aware ESC. SLA-Aware architectural styles have two unique characteristics: (1) SLA-Aware SOA applications require a set of SLM capacities from both service consumers and service providers; (2) Processing of non-functional requests (SLAs – performance, dynamic scalability, availability, etc.) of services are considered as the first-class capacity and are executed before functional operations of service. In this way, the service providers are required to provide not only functional services but also the QoS to service consumers. Capacity is the key requirement for a family of systems, for example, real-time online trading systems and online travel reservation systems. Examples include cloud services such as Amazon web services EC2 and S3, which require higher performance, availability, and dynamic scalability for satisfying the service consumers (business customers or their applications). Customers can get services and the corresponding QoS, such as performance, availability and price, based on the SLA.

To enable the dynamic SLA and SLM in a traditional ESOA stack, representing the SLA in a standard way is important. We have introduced several standard ways for defining the SLA in machine-processable languages, such as WS-Agreement, WSLA language, and WSOL. Most of the SLA languages are built on XML language. They support the SLA lifecycle in that they build, negotiate, execute, and terminate through SLA-Aware SOA management such as dynamic SLM, SLA-Aware middleware, and broker.

We define SLA-Aware ESC as an architectural style in this chapter. The primary advantage of viewing and defining SLA-Aware ESC as architectural styles is an abstraction of the common structure. Constraints of and behavior of a family of ESC systems such as ECSA-SLA style systems, and defining general design principles for the family of enterprise architectures, are other advantages. The design principles of SLA-Aware ESC systems are discussed through specifying our SLA-Aware ESOA or ECSA formula. The principles include:

- Make SLA management and QoS the first-class consideration.
- Represent SLA in standard machine-processable language.
- Manage SLA between service consumer and service provider through a dynamic SLM.
- Enable and execute SLA-based QoS operations ahead of service functional operations.
- Manage SLA between enterprise services and ESOA/ECSA infrastructure providers by SLA-Aware SOA management which includes SLA monitoring, SLA control, SLA execution, dynamic reconfiguration, and SLA lifecycle management. The SLA-Aware SOA management also supports SLA-based dynamic resource management, service provisioning, subscription and classification (rating or pricing).
- Build SLA-Aware SOA processes and workflows by end-to-end SLA management.
- Adopt autonomic service computing: Self-management, self-service, self-configuration and self-error handling and recovering.
- Adopt automated and adaptive service computing.

CHAPTER 7

ANALYSIS AND EVALUATION

We have specified and described ESOA and ECSA enterprise architectural styles and their substyles based on the proposed ontology-based modeling framework from Chapter 3 to Chapter 6 in the dissertation. The ESOA architectural style has been applied to guide the design and development of modern enterprise IT systems since 2003. The new ECSA architectural style is becoming an important design and development guideline for building more cost-effective enterprise IT systems. This chapter (1) gives further analysis on ESOA as well as ECSA; (2) describes their extension, refinement and instantiation; (3) defines the instances of ESOA and ECSA; (4) evaluates them by evaluating their instances; (5) discusses the lessons learned from applying ESOA and ECSA styles in enterprises.

7.1 Analyzing ESOA and ECSA Styles

ECSA style is defined as a combination of ESOA and ECC in Chapter 5. Here, we give further analysis of the ECSA style based on [168][207] and the framework defined in Chapter 3.

7.1.1 Checking Style Consistency

Based on [168], we should show that the two styles ESOA and ECC are conflict-free, that is, semantically no contradictions should occur. Let us assume that $ESOA^I$ and ECC^I are interpretations of ESOA and ECC, respectively. Then, it is necessary to show that $ESOA^I \cap ECC^I \neq \phi$. Obviously, this is true because:

- Both share the Service concept and the *ECC* style extends some of the concepts from *ESOA*. We have shown this in section 5.3.2.
- $SC^{i}_{ESOA} \cap SC^{i}_{ECC} \neq \phi$, $SI^{i}_{ESOA} \cap SI^{i}_{ECC} \neq \phi$, $SM^{i}_{ESOA} \cap SM^{i}_{ECC} \neq \phi$, $SP^{i}_{ESOA} \cap SP^{i}_{ECC} \neq \phi$, $SQ^{i}_{ESOA} \cap SQ^{i}_{ECC} \neq \phi$.

Both styles are complementary with each other and can coexist in an enterprise architecture. Most enterprises are moving to a hybrid cloud architecture in which two styles are applied for designing the next generation enterprise architectures. The style consistency can be checked by another criterion – "A style is consistent if there exists at least one architectural configuration that conforms to the style" [110]. This means that a style is consistent if there is at least one instance of the style. We will show that Amazon.com is an instance of the ECSA style in Subsection 7.4.

7.1.2 Checking Style Extension

ESOA defined in Chapter 4 is a top level style. It is extendable to different substyles [203][204]. Figure 4.4 depicts the hierarchy of its main substyles. ECSA defined in Chapter 5 is also the top level style which includes abstract service types and other architectural types specified in Section 5.3. It can be extended. We have introduced a notation $\lhd$ to denote the style extension relationship [206]. We have defined *ESOA* $\lhd$ *ESOA-SLA, ECSA* $\lhd$ *ECSA-SLA* in [206] and Chapter 6, in which

$$ESOA\text{-}SLA = \langle S_{SLA}, C_{SLA}, D_{SLA}, S_{SLA}I, S_{SLA}M, S_{SLA}P, S_{SLA}Q \rangle,$$

$$ECSA\text{-}SLA = \langle S_{SLA}, C_{SLA}, D_{SLA}, S_{SLA}I, S_{SLA}M, S_{SLA}P, S_{SLA}Q, S_{SLA}D \rangle.$$

The following Figure 7.1 describes the hierarchy relationship of SLA-Aware ESOA and ECSA defined in the dissertation:

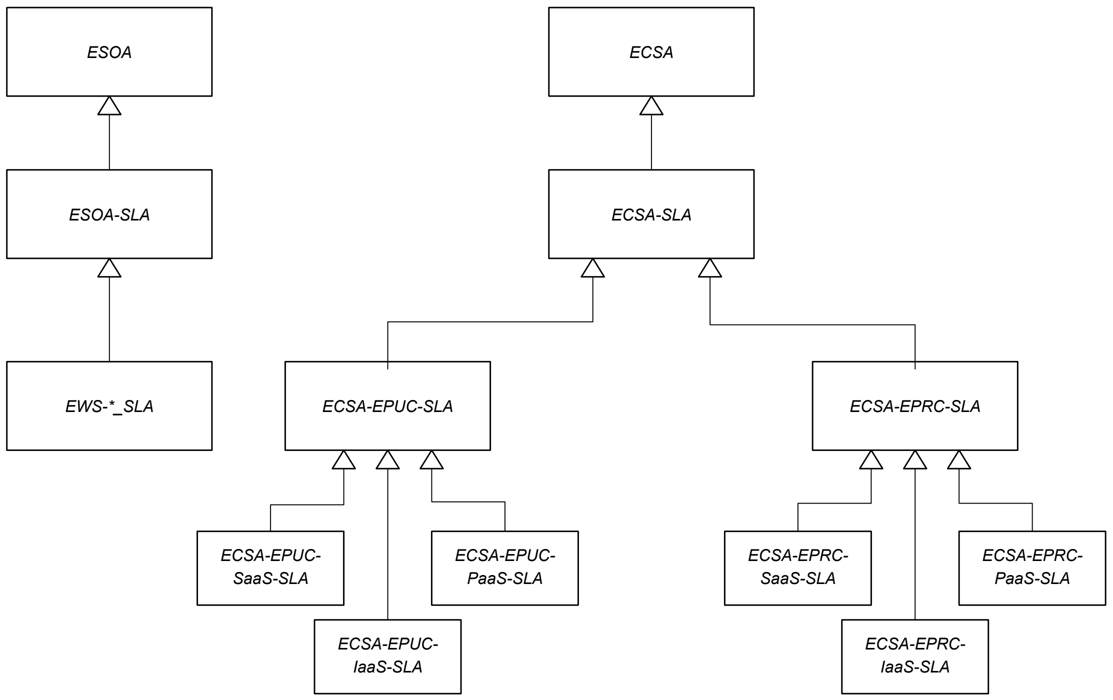


Figure 7.1. SLA-Aware ESOA and ECSA Architectural Styles Family

Moreover, enterprise private cloud *ECSA-EPRC*, enterprise service-oriented public cloud *ECSA-EPUC,* and enterprise service-oriented hybrid cloud *ECSA-EHYC* can all be defined as the sub-styles of ECSA. Therefore:

$ECSA \lhd ECSA\text{-}EPRC$

$ECSA \lhd ECSA\text{-}EPUC$

$ECSA \lhd ECSA\text{-}EHYC$

The overall hierarchy of ECSA style and its substyles is depicted as shown in the following Figure 7.2:

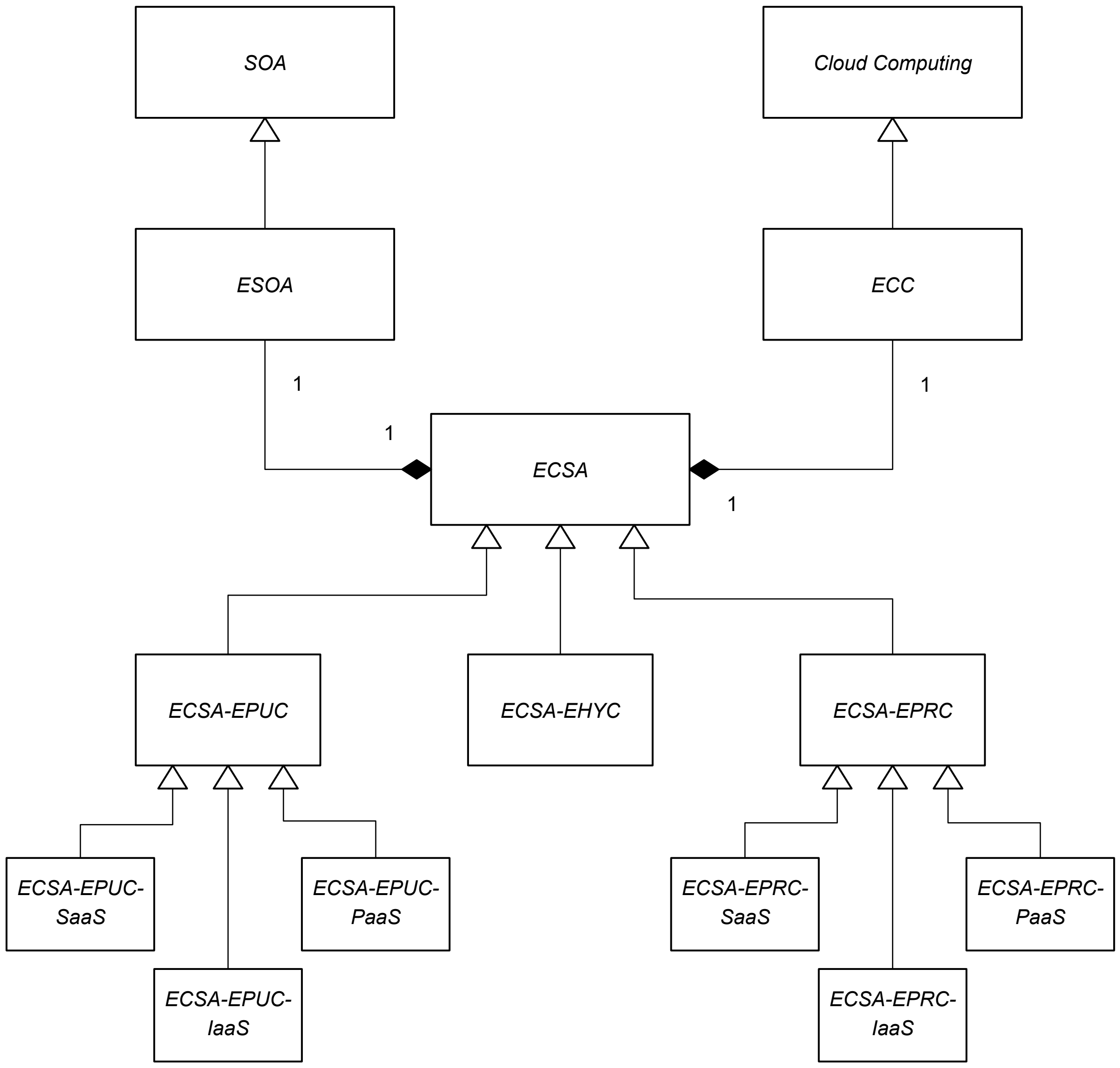


Figure 7.2. ECSA Architectural Styles Family

In the next two sections, we show the style extension as refinement of its top level style. Without loss of generality, we take *ECSA-EPRC and ECSA-EPUC* as examples.

7.1.3 Private Cloud as a Refinement of ESOA

The private cloud defined here is an architectural style – enterprise private cloud service computing, referred to as *ECSA-EPRC*, is a refinement of the *ESOA* style. From (5.5) in Chapter 5,

$ESOA=\langle \Sigma_{ESOA}, \Phi_{ESOA} \rangle$.

Also, we know that the traditional service infrastructure (*SI*) is not elastic and really dynamic, which means the topology and computing resources in traditional infrastructure are not changing with time and based on demand. This can be enhanced with a private IaaS:

IaaSSpec= $\langle \Sigma_{IaaS}, \Phi_{IaaS} \rangle$,

which is an internal IaaS specification with elastic and dynamic capacity of cloud IaaS. Obviously, $\Sigma_{ESOA} \cap \Sigma_{IaaS} = \phi$, so

$$\begin{aligned} ECSA\text{-}EPRC &= ESOA \oplus \langle \Sigma_{IaaS}, \Phi_{IaaS} \rangle \\ &= \langle \Sigma_{ESOA} + \Sigma_{IaaS}, \Phi_{ESOA} + \Phi_{IaaS} \rangle . \end{aligned} \tag{7.1}$$

Equation (7.1) means that the dynamic infrastructure concept, structure, and description are added into the traditional ESOA style. Since *ECSA-EPRC* is a refinement of *ESOA*, it can be a sub-style of *ECSA*, that is:

$ECSA \lhd ECSA\text{-}EPRC$.

7.1.4 Service-Oriented Public Cloud as Refinement of ECC

Enterprise cloud computing as a new distributed computing style is immature and faces many challenges, such as security and service governance. Relatively mature ESOA, such as its standards, service management, and process can help cloud computing to reduce its risks and adaptation by enterprises. From the architectural style point of view, the immature ECC needs to be refined in terms of mature ESOA style. Let enterprise public cloud service computing style = *ECSA-EPUC*. We show that it is a refinement of ECC with ESOA. From (5.6) in Chapter 5,

$ECC=\langle \Sigma_{ECC}, \Phi_{ECC} \rangle$.

Let

$$\text{SomeESOASpec}=\langle \Sigma_{SM}, \Phi_{SM}\rangle,$$

where Σ_{SM} is a part of some specific enterprise server management concepts, such as policy management and enforcement from the ESOA style. Clearly, $\Sigma_{ECC} \cap \Sigma_{SM} = \phi$, so we have:

$$\begin{aligned} \mathit{ECSA\text{-}EPUC} &= \mathit{ECC} \oplus \langle \Sigma_{SM}, \Phi_{SM}\rangle \\ &= \langle \Sigma_{ECC} + \Sigma_{SM}, \Phi_{ECC} + \Phi_{SM}\rangle. \end{aligned} \tag{7.2}$$

From (7.2), we can see that some of the specific ESOA service management (or governance) concepts, structures, and their descriptions are added to the ECC style, so that *ECSA-EPUC* is a refinement of ECC with ESOA. That is:

$\mathit{ECSA} \lhd \mathit{ECSA\text{-}EPUC}$

7.2 Instance of ESOA and Case Studies

This section analyzes several enterprise systems in order to check whether they meet the ESOA model specified in Chapter 4. For an enterprise system to meet the ESOA model requirement, the system needs to satisfy the following conditions:

(1) The structure and behavior of EA satisfy the enterprise service orientation formula (5.1).

(2) EA is an instance of any of the following styles:

- EWS-* style;
- EEDA style;
- EWOA style;
- ECBS style;
- EGSA style and
- Hybrid style.

(3) The core services in EA satisfy the service properties defined in S_p.

We define that an Enterprise Architecture (EA) is an instance of ESOA if it satisfies the above conditions. An EA evaluation form is designed for evaluating the following different enterprise architectures.

This section chooses five concrete EA for case studies. Section 7.2.1 is a traditional EAI integration architecture based on Gartner Research [109]. We show that it is not an instance of ESOA. Section 7.2.2 is a typical EA of ESOA hybrid style with EWS-* and ECBS. Section 7.2.3 shows that an EA, similar to the EA in Section 7.2.2, is not an instance of ESOA if it lacks SOA management and security components in design. Section 0 shows that an EA based on open source ESOA is an instance of ESOA. This will help architects to make decisions on choosing open source as ESOA building blocks. Finally, Section 0 shows that an EA built on IBM ESOA products can be an instance of ESOA. It can help architects to make the right decision on evaluating and selecting vendor's products when designing ESOA systems.

7.2.1 Traditional EAI

Figure 7.3 depicts a traditional EAI - Spaghetti-like enterprise architecture [109]. It is easy to see that the enterprise architecture does not satisfy conditions (1), (2) and (3). Thus, it is not an ESOA style architecture. Table 7.1 specifies why traditional EAI is not an instance of ESOA.

Table 7.1. Evaluation Form for Traditional EAI

EA Evaluation Form		
Condition	Satisfy	Rationale
(1)	No	Not service oriented, but application oriented
(2)	No	• Traditional EAI style. • Complex application infrastructure
(3)	*No*	• *Tight coupling between applications.* • *Hard to change and adapt to business needs.* • *Poor scalability.* • *Security is not well addressed.*

Traditional EAI was developed to solve enterprise integration, such as application integration and business to business integration (B2B). However, it failed to deliver its promise and resolve some business issues. The lessons learned from traditional EAI are as follows.

- A data centric EAI approach is not good enough for enterprise architecture which needs to serve complicated business processes. That is why the SOA process is one of the main parts in the ESOA style.
- Tight coupling leads to hard to maintain enterprise systems. Loose coupling is a way to increase business agility.
- ESOA is a better way for enterprise application integration.

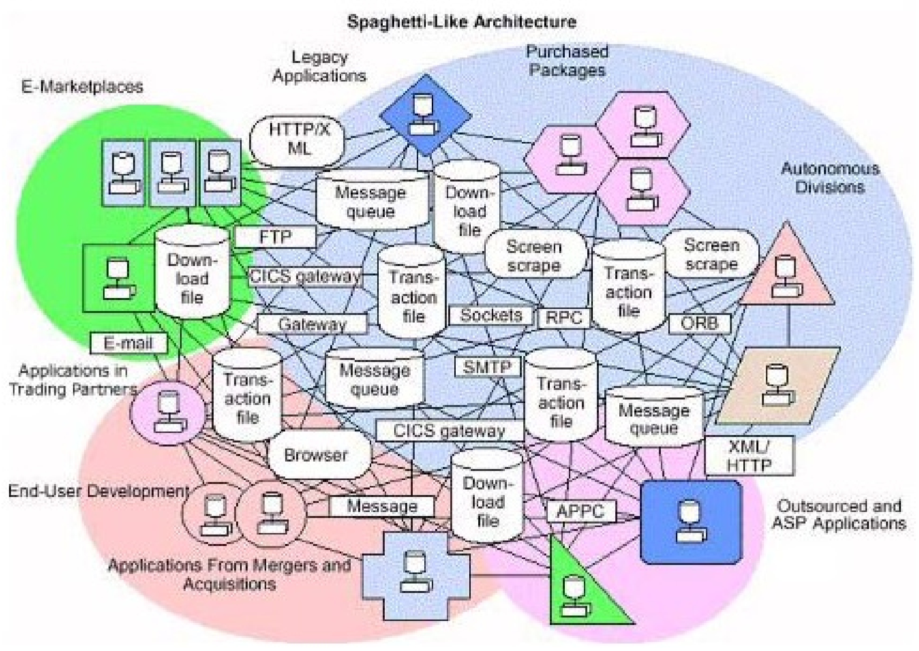


Figure 7.3. Traditional EAI – Spaghetti-like Architecture

7.2.2 Hybrid ESOA System

Figure 7.4 describes the typical concrete enterprise architecture (CEA). This example assumes that stateless Enterprise Java Bean (EJB) Version 3 is chosen as the internal business service component for internal customers and its wrapper of SOAP-based web service as the external business service for external customers; either Weblogic 10.3 or WebSphere 6.1 is chosen as the application server; either BEA System's Aqualogic Service Bus or IBM ESB is chosen as the Enterprise Service Bus (ESB); and Apache Tuscany is chosen as the service component process engine. The EA is adopted by some enterprises whose traditional EAI is based on SUN J2EE and has many existing J2EE applications and services. Table 7.2 shows the EA is an instance of ESOA with hybrid style.

Table 7.2. Evaluation Form of a Hybrid ESOA System

EA Evaluation Form		
Condition	Satisfy	Rationale
(1)	Yes	$\boldsymbol{EA_{CEA}}$= $\langle S_{CEA}, C_{CEA}, D_{CEA}, SI_{CEA}, SM_{CEA}, SP_{CEA}, SQ_{CEA} \rangle$
(2)	Yes	Hybrid of EWS-* style and ECBS style • ESB for exposing web server interfaces to outside service consumers: EWS-* style • Application server for exposing component-based server interfaces to inside service consumers: ECBS style
(3)	Yes	Detail rationalization is specified in last part of the subsection.

In Table 7.2, the parts of the EA can be defined in detail as follows:

$\boldsymbol{S_{CEA}}$ = { s | s is a stateless Java session EJB or a Message-Driven EJB or SOAP-Based Web Service}

$\boldsymbol{C_{CEA}}$ = {Inside Services Clients, Outside Services Clients}

$\boldsymbol{D_{CEA}}$= {Server metadata, EJB metadata, Web server configuration data, Application server configuration data, ESB configuration data}

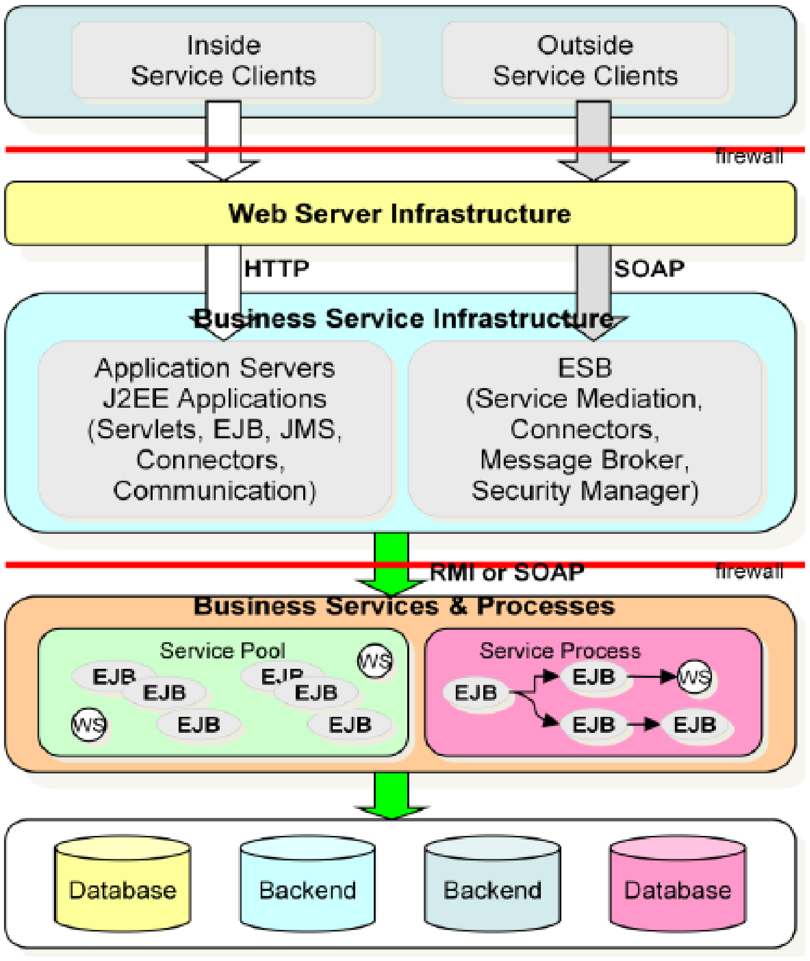
$\boldsymbol{SI_{CEA}}$ = {Web server infrastructure, Application server infrastructure, ESB}

$\boldsymbol{SM_{CEA}}$ = {Application Server Management, ESB Management, Monitors, Security Management, Network Management}

$\boldsymbol{SP}_{CEA}$ = {EJB-based component workflows} ∪ {SCA-based service process}

$\boldsymbol{SQ}_{CEA}$ = {Performance, Scalability, Reusability, Reliability, Security,

Maintainability} ∪ S_p


Inside Service Clients
Outside Service Clients
firewall
Web Server Infrastructure
HTTP
SOAP
Business Service Infrastructure
Application Servers J2EE Applications (Servlets, EJB, JMS, Connectors, Communication)
ESB (Service Mediation, Connectors, Message Broker, Security Manager)
RMI or SOAP
firewall
Business Services & Processes
Service Pool
EJB
EJB
EJB
EJB
EJB
EJB
WS
WS
Service Process
EJB
EJB
EJB
EJB
EJB
WS
Database
Backend
Backend
Database
Figure 7.4. An Enterprise Architecture

One just needs to verify if the EJB-based core services satisfy condition (3).

- Standard Service Contracts – stateless session EJB uses remote and local EJB interfaces of the component-based contracts. The message-driven EJB provides an onMessage interface for asynchronous client's interaction. If all EJB services are designed with enterprise standard interfaces, then the core EJB services have standardized service contracts. We will discuss how to specify standard EJB interface in an EJB-based service inventory in an enterprise in our future work.

- Reusability – unlike a public web service, such as the *weather* service, which has universal reusability, EJB is a reusable Java component in enterprise business domain.
- Relative Autonomy – An EJB can perform its work independently of most of the other components or applications. However, an EJB must be executed inside an EJB container. Therefore, EJB has service-level and contractual autonomy, rather than pure autonomy.
- Statelessness – both stateless sessions EJB and message-driven EJB are stateless.
- Discoverability - EJB uses JNDI (Java Naming Directory Interface) for locating home interfaces, business methods, and metadata. Therefore, it can be dynamically discoverable.
- Relatively Loose coupling – in the concrete EA, the core services are wrapped by public services interfaces exposed to outside service consumers. From the view of an outside client, they are loosely coupled. However, they expose the services to inside service consumers because EJB plus Remote Method Invocation (RMI) are coupled at both its Java language and platform. The message-driven EJB supports loose coupling at the service-level by its asynchronous messaging. The stateless session EJB (before EJB 3.0) is also tightly coupled with its clients to a certain degree through RMI stub. However, the coupling has been improved by EJB 3.0. Moreover, the dependency injection supported by EJB 3.x also greatly reduces the coupling between EJB components and infrastructure. By adopting group services versioning and the “unit of deployment” for services and service consumers, the

tight coupling will benefit execution performance. Therefore, one can say that the design of core services of the EA achieves a certain degree of loose coupling.

- Abstraction – EJB specification abstracts the non-essential service information through several types of meta abstraction which are defined in ejb-jar.xml for each EJB. The annotation metadata model is introduced in the newest EJB 3.0. The container managing the transaction behaviors is hidden by the tag <container-transaction>. The QoS policy, such as performance (pool size) and security (run-as-identity-principle), can be defined in EJB meta XML files.
- Composability – Software component is composable by definition [123]. EJB is a Java-based component. Therefore, in nature, it can be composable with other components, such as other EJBs, for executing a business process, such as workflows and transactions.

In conclusion, the EA described in Figure 7.4 is an instance of hybrid ESOA. From the case study, it is clear that the hybrid style's core style is ECBS based on components (such as EJB and .NET). It may have tightly coupled API and, thus, may have to be tightly controlled. Thus, the style is still inflexible and hard to scale. Although the component services are only used for internal consumers and other technologies, such as versioning and the "unit of deployment", can reduce the bad effort from tight coupling, the maintainability and agility are impacted. The adoption of the style is based on performance consideration and existing EAI systems for some enterprises.

7.2.3 An Incomplete ESOA System

If two important components – security and monitor - are removed from the EA in Figure 7.4 by design, the EA becomes an incomplete ESOA system. Table 7.3 specifies the incompleteness of the EA. The architecture does not satisfy the architecture quality attributes reliability and performance (part of QoS policy) defined in (4.2) and Section 4.5.7. Therefore, the architecture does not satisfy ESOA QoS policy.

If the core services use stateful EJB, then they also violate the "statelessness" - one of the service properties defined in Section 4.5.1. Therefore, the EA is not a complete ESOA system. However, this system can be changed so that it will be qualified as an ESOA by adding those missing components and attributes.

From the case study, the proposed ESOA style model is helpful not only in understanding ESOA, but also in analyzing and evaluating SOA enterprise architecture, and finding the missing parts in an ESOA system. Moreover, the next section shows that the ESOA style model and its instance enterprise ESOA style architecture defined in this chapter can help enterprise architects to make decision on adopting the right open source products in building ESOA architecture.

Table 7.3. Evaluation of the Incomplete ESOA System

EA Evaluation Form		
Condition	Satisfy	Rationale
(1)	Partial	Without security and monitoring in enterprise architecture, its infrastructure and services as well as their processes do not satisfy the SOA Management (SM) defined in (4.2) and Section 4.5.5.
(2)	Yes	Hybrid style
(3)	Partial	• Moreover without security manager, the architecture does not satisfy the architecture security attribute; • Without system monitors in an enterprise architecture, there is no way to detect failures of service and infrastructure and to measure performance of services and processes.

7.2.4 FUSE ESB for ESOA System

This subsection shows that the open source FUSE ESB products based on Apache ServiceMix can be used for building ESOA architecture by using the model and style analysis proposed in this chapter. Table 7.4 specifies the characteristics of the ServiceMix-based architecture (SMA).

Based on SMA as shown in Figure 7.5, one can define the parts of SMA in Table 7.4:

$\boldsymbol{S_{SMA}}$ = { s | s is a stateless Java session EJB or a Message-Driven EJB or SOAP-based Web Service, RESTful Web Service}

$\boldsymbol{C_{SMA}}$ = {Internal Services Clients, External Services Clients}

$\boldsymbol{D_{SMA}}$ = {Web server infrastructure data, Application server configuration data, ESB configuration data, jmx.xml, file-poller-su, eip-wiretap-su, camel-persist-su, eip-cbr-su, jms-producer-su, …}

in which all *-su are configuration XML files for service unit which provide information of the service and their endpoints to the component.

$\boldsymbol{SI_{SMA}}$ = {Web server infrastructure, Application server infrastructure, ESB}

$\boldsymbol{SM_{SMA}}$ = {Service life-cycle management, Service Policy Management, Monitors, Security Management, Network Management}

$\boldsymbol{SP_{SMA}}$ = {ODE-based service process} $\cup$ {SCA-based service process}

$\boldsymbol{SQ_{SMA}}$ = {Performance, Scalability, Reusability, Reliability, Security, Transactionability, Scalability, Manageability, Interoperability} $\cup\, S_p$

Therefore, based on the evaluation of ServiceMix-based EA, such as SMA in Figure 7.5, the system is an instance of ESOA. However, because the ServiceMix supports only a subset of WS-* specifications together with limited management and SOA process (see Table 7.5), it can be used for creating agile and lightweight ESOA systems for small or middle size service orientation enterprises.

Table 7.4. Evaluation of FUSE ESB ESOA Architecture

EA Evaluation Form		
Condition	Satisfy	Rationale
(1)	Yes	The ServiceMix [179] is built based on SUN's JBI specification [210]. Its core includes the ServiceMix Enterprise Service Bus (ESB) based on JBI Normalized Message Route (NMR) and an OSGi-based ServiceMix Kernel. It is not only an ESB, but also provides a JMX-based SOA management and many other enterprise capabilities, such as ActiveMQ for supporting EDA, Apache ODE BPEL engine as a drop-in JBI component for ServiceMix. Therefore the ServiceMix-based architecture *SMA* in Figure 7.5 satisfies the service oriented formula: **EA_{SMA}**= $\langle S_{SMA}, C_{SMA}, D_{SMA}, SI_{SMA}, SM_{SMA}, SP_{SMA}, SQ_{SMA} \rangle$
(2)	Yes	ServiceMix combines the functionalities of SOA and EDA, as well as supports multiple types of services; therefore SMA is a hybrid system with • EWS-* style • EEDA style • ECBS style
(3)	Yes	• A subset of WS-* specifications are supporting through ServiceMix's binding components as listed in Table 7.5. • ServiceMix supports SOA security, reliability, manageability and transactionability through supporting WS-* specifications. Moreover it supports clustering and load balancing through multiple ServiceMix instances communication via JMS using ActiveMQ. It also can use lightweight cache binding component for improving SOA performance. • The web services in EWS-*, EEDA and ECBS styles satisfy the service properties defined in S_p .

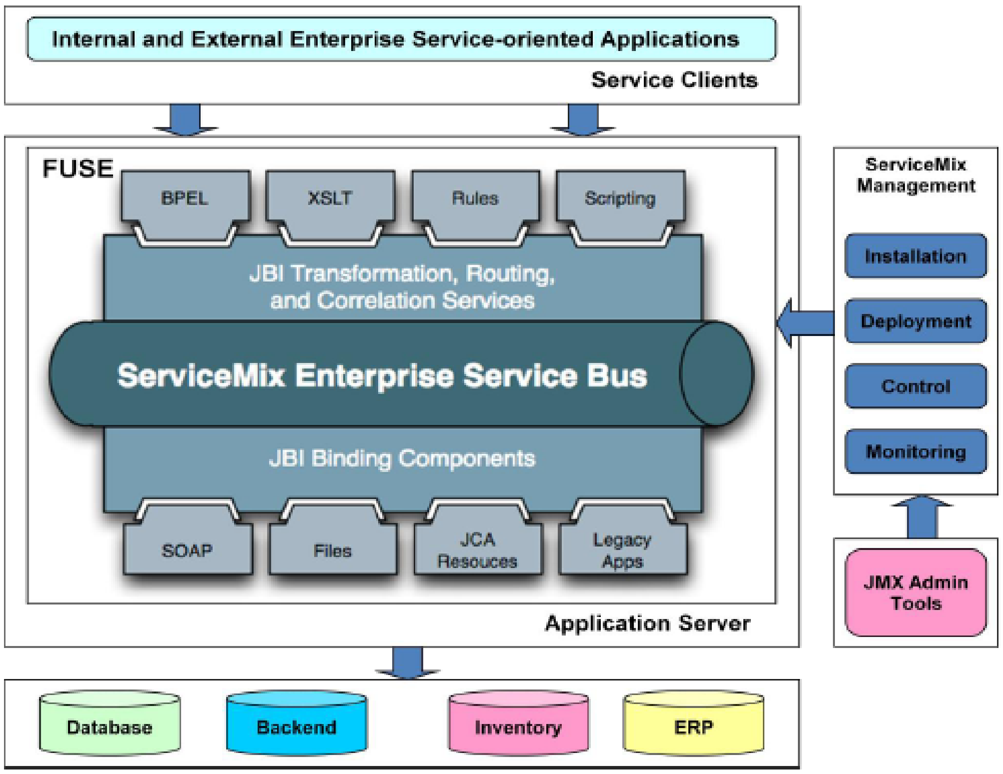


Figure 7.5. FUSE-ServiceMix for ESOA Style Architecture

Table 7.5. WS-* Specification for ServiceMix

WS-* Spec	Purpose	Supported by ServiceMix
WS-Security	Authentication, Encryption, Digital Signature	Yes, for HTTP and CXF (An open source service framework) binding components and subsequent authentication/authorization
WS-RM	Reliable Messaging	Yes, for CXF binding component
WS-Address	Addressing	Yes, for HTTP, JMS (Java Message Service) and CXF binding components
WS-Policy	Policy management	Yes, for CXF binding component
WS-Notification	Events	Yes
WS-TX	Transaction	No, though WS-TX headers can be passed through as normalized message headers for services to handle
WSDM	Management	No, JMS is used instead
WS-Management	Management	Not directly, JMX (Java Management Extensions) is used instead; a bridge from JMS to WS-Management is being developed.

7.2.5 Enterprise Systems based on IBM WebSphere

Many SOA solution providers, such as IBM, SAP, Oracle, and Microsoft, have developed their SOA models and products for building service-oriented enterprise, such as IBM WebSphere [3], SAP NetWeaver [214][41], Oracle Fusion Middleware [163] and Microsoft WCF as well as its products [22]. It is easy to show that the enterprise architecture based on any one of them is an instance of ESOA. For example, Figure 7.6 illustrates an enterprise system built on IBM SOA application architecture and is an instance of the hybrid style. Table 7.6 describes the main characteristics of IBM SOA-based application architecture.

Table 7.6. Evaluation of IBM SOA-Based Architecture

EA Evaluation Form		
Condition	Satisfy	Rationale
(1)	Yes	The EA can be represented in the enterprise service orientation formula (4.2) based on Figure 7.6 and [3]: ***EA*** $\boldsymbol{EA_{IBM}}$= $\langle S_{IBM}, C_{IBM}, D_{IBM}, SI_{IBM}, SM_{IBM}, SP_{IBM}, SQ_{IBM} \rangle$
(2)	Yes	IBM Enterprise SOA solutions and products support service loose coupling and event-driven architecture through mediation, such as ESB, and messaging, such as MQ; they also support RESTful web services through provisioning Web 2.0 Feature Pack; they also support SCA through provisioning SCA Feature Pack. Therefore an EA based on IBM ESOA can have multiple ESOA styles – EWS-*, EWOA, EEDA and ECBS.
(3)	Yes	• IBM Enterprise SOA Application Architecture is built on SOA principles. Many non-functional requirements (or SOA quality attributes) are considered when designing IBM SOA products, such as WebSphere, and other products. Therefore it can guarantee an EA built on IBM SOA solution to reach its QoS and SLA goals. Table 7.7 shows how IBM SOA solution meets the enterprise SOA quality requirements. • Moreover the services satisfy the service properties defined in S_p .

In Table 7.6, the parts of the EA can be described in detail as follows:

$\boldsymbol{S_{IBM}}$ = { s | s is a Component-based service, SOAP-based Web Service, RESTful Web Service or Event-Based Service},

$\boldsymbol{C_{IBM}}$ = {Web client, event-based client, offline client, web service client},

$\boldsymbol{D_{IBM}}$ = {Web server infrastructure data, WebSphere server configuration data, WebSphere Service metadata, WebSphere policy data, WebSphere Process Execution Rules},

$\boldsymbol{SI_{IBM}}$ = {Web server infrastructure, WebSphere infrastructure, WebSphere ESB, WebSphere MQ, WebSphere Service Registry, WebSphere Service Integration Bus},

$\boldsymbol{SM_{IBM}}$ = {IBM Service life-cycle management, IBM Service Policy Management, Tivoli Monitors, WebSphere Business Monitor,

WebSphere Security Management, Tivoli Identity Management,

IBM SOA Connectivity Management, WebSphere Process Server},

$\boldsymbol{SP_{IBM}}$ = {WS-BPEL based process} $\cup$ {SCA-based service process},

$\boldsymbol{SQ_{IBM}}$ = {Performance, Scalability, Reusability, Reliability,

Security, Transactionability, Scalability, Manageability, Testability,

Interoperability, Maintainability} $\cup\, s_p$.

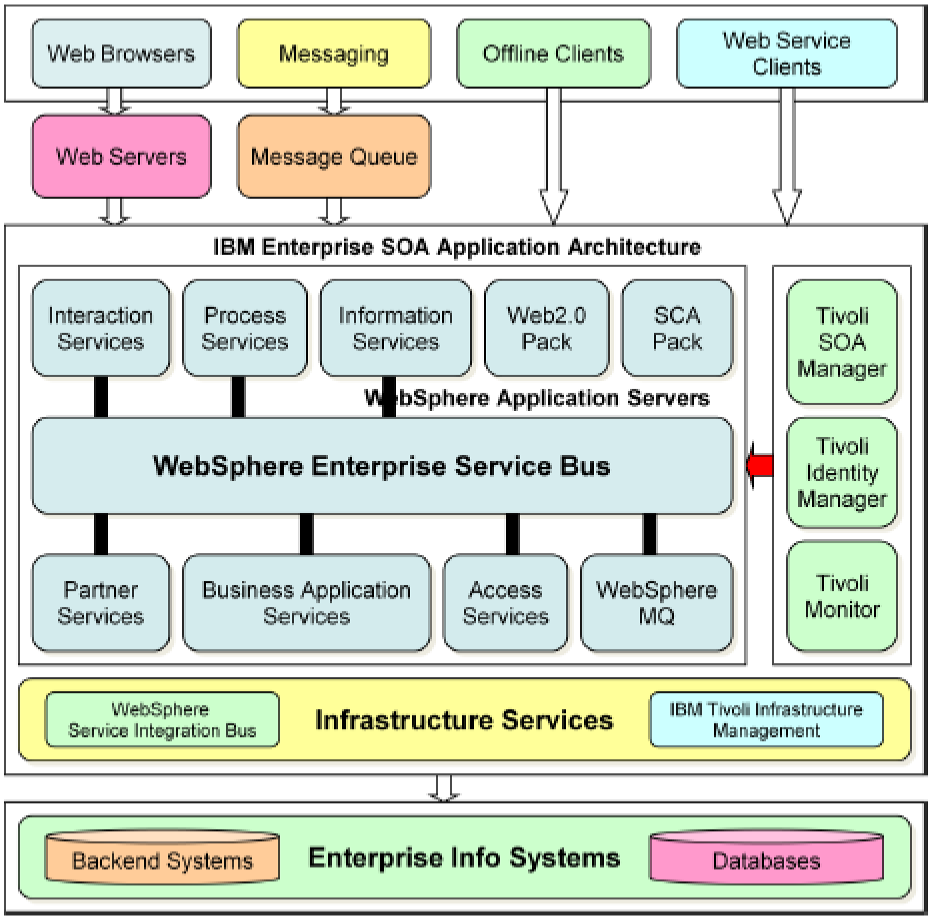


Figure 7.6. EA Built on IBM SOA Products

In conclusion, the IBM SOA-based EA is an instance of the hybrid ESOA. From the case study, it is clear that:

- IBM as an ESOA product vendor can provide a package of products from services, SOA data, service infrastructure, service management to SOA quality of service. Therefore, IBM can be chosen as a vendor for building an ESOA system.
- The advantage of a single vendor approach, such as IBM, for building ESOA systems is that it is easy to manage and maintain the systems because of high comparability among ESOA parts. However, a single vendor may not guarantee that every product is the best in the market in pricing and quality, so some large enterprises prefer building ESOA systems based on diversifying vendor products, such as choosing weblogic as the application server and Wily product as the application monitor.

7.3 Instance of ECSA and Case Studies

If a concrete EA (CEA) satisfies the following conditions, we call the CEA an instance of ECSA:

(1) The CEA can be described by the ECSA ontology (5.2)-(5.4).

(2) The CEA satisfies any one of the following cloud extensions of ESOA or any of the enterprise cloud with service orientation:

   - Enterprise Private Cloud (EPRC);
   - Enterprise Public Cloud (EPUC);
   - Enterprise Hybrid Cloud (EHYC).

(3) The cloud extensions should meet the quality attributes and the public cloud services should satisfy the properties defined in Section 5.4.7.

Table 7.7. IBM SOA Quality Attributes

SOA Quality Attributes	IBM Solutions & Products
Performance	WebSphere DataPower, Performance Monitoring Provisioning enhancements of Web Service and EJB 3.0
Scalability	WebSphere eXtreme Scale
Security	Tivoli Identity Manager WebSphere Security Management WebSphere ESB
Interoperability	WebSphere Interaction Service WebSphere MQ WebSphere ESB
Reliability	WebSphere eXtreme Scale Workload management Reliable messaging
Availability	Failover High-availability clustering On-demand routing Workload management controller
Transactionability	WebSphere Process Service Workflow management
Manageability	WebSphere Informantion service Tivoli Monitoring Tivoli Infratructure Management WebSphere ESB
Maintainability	WebSphere Adminitraction Services IBM SOA lifecycle management
Testability	WebSphere Test Envirnmrnt & tools

7.3.1 ECSA-EPRC Style Instance

Figure 7.7 describes an ESOA CEA with dynamic infrastructure. We show that the CEA is an instance of the ECSA style by the evaluation form shown in Table 7.8.

Table 7.8. Evaluating SOA EA with EPRC

CEA Evaluation Form		
Condition	Satisfy	Rationale
(1)	Yes	Service oriented infrastructure is cloud enabled. It can be described by the (5.2)-(5.4). $CEA_{EPRC} = \langle S_{EPRC}, C_{EPRC}, D_{EPRC}, SI_{EPRC}, SM_{EPRC}, SP_{EPRC}, SQ_{EPRC}, SD_{EPRC} \rangle$
(2)	Yes	It is an EPRC.
(3)	Yes	Infrastructure satisfies cloud quality attributes, such as elasticity, flexibility.

It is easy to specify its ESOA architectural components [204]. We just need to show that it also satisfies EPRC and some specific cloud quality attributes. By the CEA, $SD_{EPRC} \neq \emptyset$. Since it delivers IaaS-type of cloud service, either by internal cloud provider or by third party through VPC, it is a private cloud. Therefore, $SD^{II}_{EPRC} \neq \emptyset$ and $SD^{III}_{EPRC} \neq \emptyset$. Moreover, the CEA satisfies a subset of the cloud quality attributes, such as elastic scalability and flexibility.

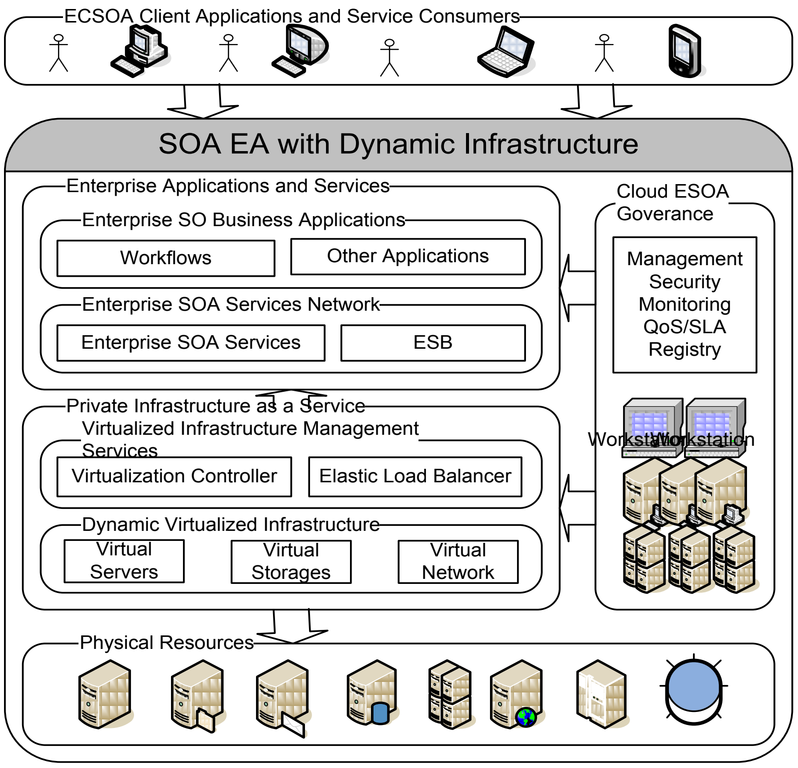

ECSOA Client Applications and Service Consumers
SOA EA with Dynamic Infrastructure
Enterprise Applications and Services
Enterprise SO Business Applications
Workflows
Other Applications
Enterprise SOA Services Network
Enterprise SOA Services
ESB
Cloud ESOA Goverance
Management
Security
Monitoring
QoS/SLA
Registry
Private Infrastructure as a Service
Virtualized Infrastructure Management Services
Virtualization Controller
Elastic Load Balancer
Workstation
Workstation
Dynamic Virtualized Infrastructure
Virtual Servers
Virtual Storages
Virtual Network
Physical Resources

Figure 7.7. ESOA Style EA with Dynamic Infrastructure

Many existing service oriented enterprises are moving to this kind of ECSA style for building green IT and smart SOA datacenters.

7.3.2 ECSA-EPUC Style Instance

Figure 7.8 describes a CEA built on the IBM cloud and SOA architecture, which can provide public cloud services. Table 7.9 shows that the EA is an instance of ECSA.

Table 7.9. Evaluating the CEA Built on IBM Cloud

CEA Evaluation Form		
Condition	Satisfy	Rationale
(1)	Yes	Service oriented and cloud enabled. It can be described by the (5.2)-(5.4). $CEA_{EPUC} = \langle S_{EPUC}, C_{EPUC}, D_{EPUC}, SI_{EPUC}, SM_{EPUC}, SP_{EPUC}, SQ_{EPUC}, SD_{EPUC} \rangle$
(2)	Yes	It is an EPUC
(3)	Yes	It satisfies cloud quality attributes, such as elasticity, flexibility.

In Table 7.9

S_{EPUC} = {traditional services} ∪ {public cloud services},

C_{EPUC} = {traditional service consumers} ∪ {public cloud service consumers},

$D_{EPUC} = D^{I}_{EPUC} \cup D^{II}_{EPUC}$,

where D^{I}_{EPUC} is SOA data defined in [202][204] and

D^{II}_{EPUC} = {IBM cloud metadata, cloud SLA data, cloud QoS data,

IBM virtualization metadata, cloud service registry data},

SI_{EPUC} = {virtualized servers, virtualized storage, virtualized network} ∪

{physical servers, physical storage, physical network},

SM_{EPUC} = {Tivoli User Request Manager, Self-service portal,

service lifecycle manager, Tivoli security manager, performance manager,

Tivoli monitoring, Usage Accounting service, Provisioning service,

workflow management, Virtualization management,

policy management, SLA management},

SP_{EPUC} = {traditional ESOA business processes} ∪ {cloud business process} ∪

{cloud virtualization orchestration},

SQ_{EPUC} = {performance, elastic scalability, availability, security, accountability, visibility},

SD_{EPUC} = {Prc, HyC} ∪ {SaaS, PaaS, IaaS}.

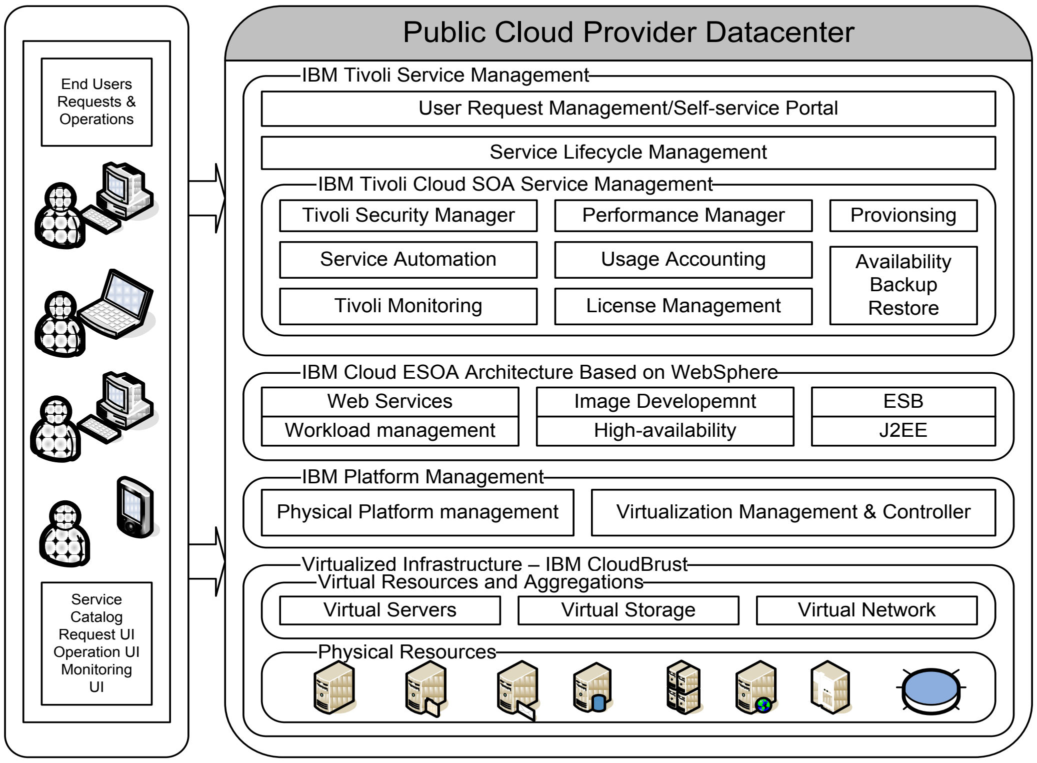

Public Cloud Provider Datacenter
End Users Requests & Operations
Service Catalog Request UI Operation UI Monitoring UI
IBM Tivoli Service Management
User Request Management/Self-service Portal
Service Lifecycle Management
IBM Tivoli Cloud SOA Service Management
Tivoli Security Manager
Performance Manager
Provionsing
Service Automation
Usage Accounting
Availability Backup Restore
Tivoli Monitoring
License Management
IBM Cloud ESOA Architecture Based on WebSphere
Web Services
Image Developemnt
ESB
Workload management
High-availability
J2EE
IBM Platform Management
Physical Platform management
Virtualization Management & Controller
Virtualized Infrastructure – IBM CloudBrust
Virtual Resources and Aggregations
Virtual Servers
Virtual Storage
Virtual Network
Physical Resources

Figure 7.8. Public Cloud Built on IBM Cloud

7.3.3 ECSA-EHYC Style Instance and Z Cloud

An ESOA style EA with EHYC in enterprise A is shown in Figure 7.9. We show that it is an instance of ECSA in the evaluation form shown in Table 7.10.

The following Figure 7.10 describes the EHYC style instance Zynga hybrid cloud service architecture. Zynga.com is an online game service provider which serves about 250 million active users a month, with 90 millon of them coming from its CityVille addition [20]. It is using

Table 7.10. Evaluating ESOA Style EA with EHYC

CEA Evaluation Form		
Condition	Satisfy	Rationale
(1)	Yes	Service oriented and cloud enabled. It can be described by the (5.2)-(5.4). $CEA_{EHYC} = \langle S_{EHYC}, C_{EHYC}, D_{EHYC}, SI_{EHYC}, SM_{EHYC}, SP_{EHYC}, SQ_{EHYC}, SD_{EHYC} \rangle$
(2)	Yes	It is an EHYC.
(3)	Yes	It satisfies cloud quality attributes, such as elasticity, flexibility.

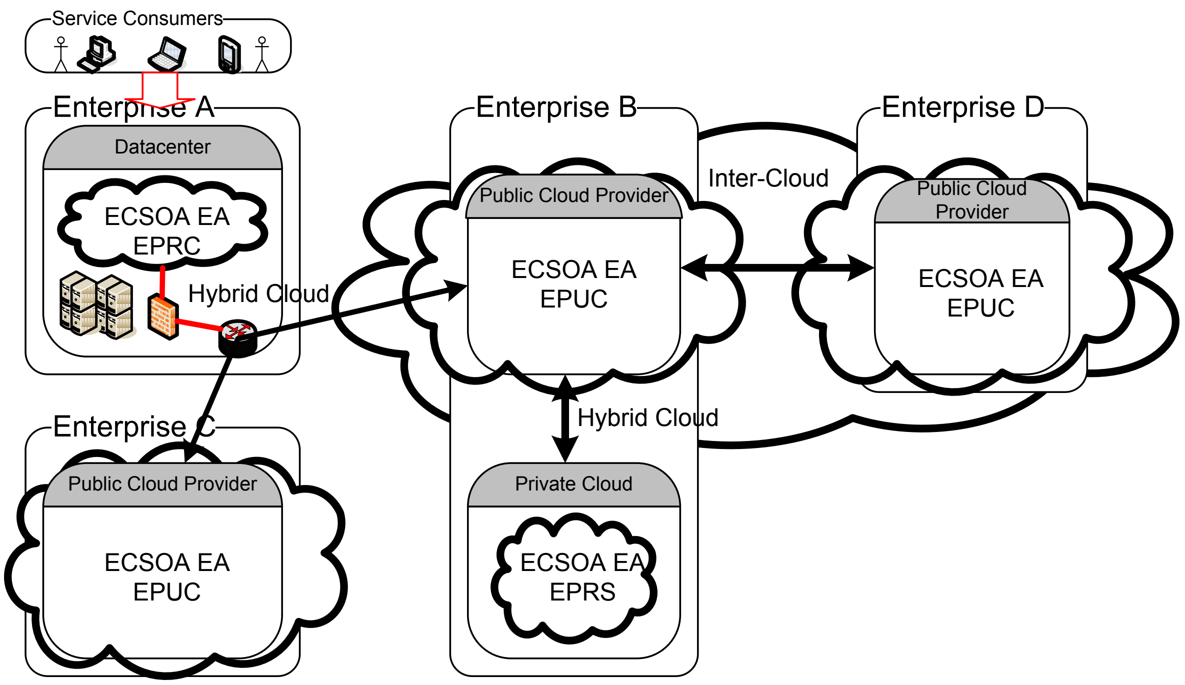


Figure 7.9. ESOA Style EA with EHYC

Amazon EC2 service, but its business was not impacted by the Amazon 12 hours big EC2 outage on 04/21/2011[20]. Our quality ontology can give us the reasons why Zynga is not impacted by EC2 outage. Based on its architecture, its high scalability is obtained from both scaleOut and scaleIntoCloud (EC2). However, its Availability management is based on failover and fail-protect between its Z Cloud datacenter and A Cloud in Amazon datacenter. When Amazon EC2 is down, it is easy to route Zynga's games' and users' traffic to its private Z Cloud with high availability. The lessons learned from Zynga is (1) design should consider multiple failure cases; (2) failover in a single datacenter is not enough; (3) we should consider using multiple ways to prevent single point failures, which include failures across datacenters and clouds. Therefore, hybrid cloud service architectures allow enterprises to gain both high scalability and availability.

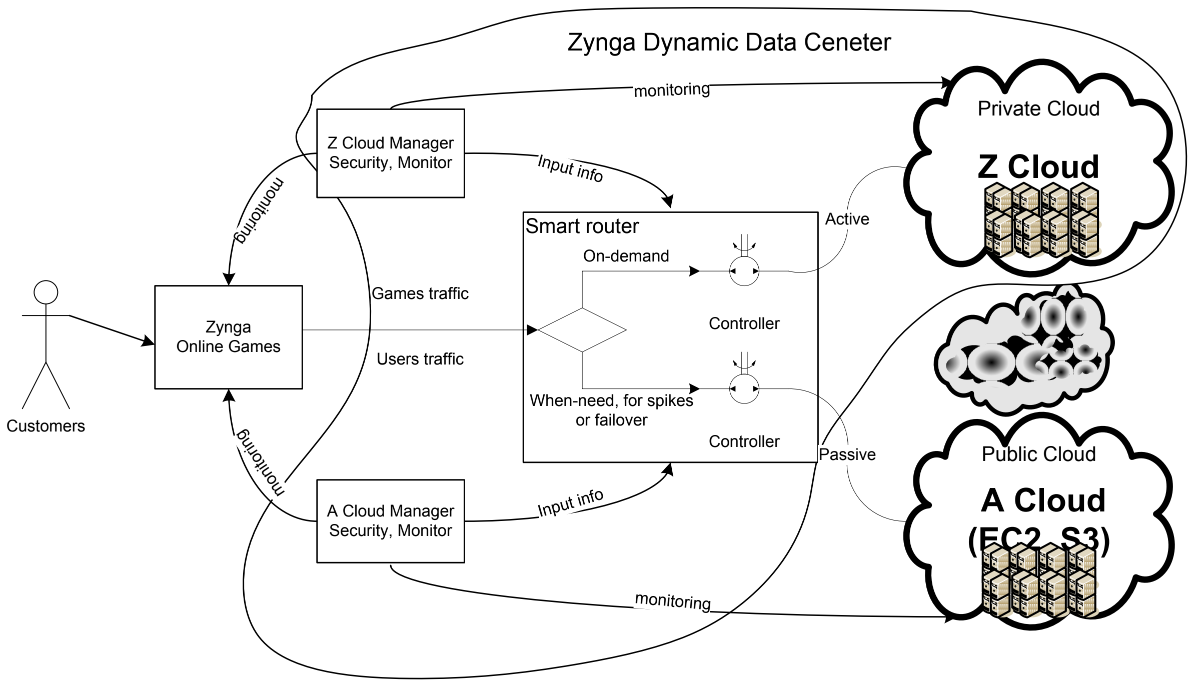


Figure 7.10. Zynga Hybrid Cloud Service Architecture

We have evaluated several typical EA and provided the guidelines to check if an EA is an instance of ECSA. If a cloud-centric architecture does not adopt the SOA, which means that it does not meet the condition (2), then the cloud architecture is not an instance of ECSA.

7.3.4 Amazon Cloud Architecture (ACA)

ACA is an instance of ECSA shown in this subsection. Figure 7.11 is a simplified Amazon Cloud Architecture. Let

$$S_{ACA} = \{LBS, EC2, S3, SQS, SDB, EBS, BS\}, \qquad (7.1)$$

$$C_{ACA} = \{C_{EC2}, C_{S3}, C_{SQS}, C_{SDB}\}, \qquad (7.2)$$

SI_{ACA}={WEB,RT,ELB,AVM,ASP,VPN,DS}, (7.3)

SM_{ACA}={CWM,ERM,BM}, (7.4)

SP_{ACA}={KVS, PWF}, (7.5)

SD_{ACA}={{PrC, PuC, VPC},{IaaS}}, (7.6)

In (7.1)

LBS=Load Balance Service

EC2=Amazon Elastic Compute Cloud Service

S3=Amazon Simple Storage Service

SQS=Amazon Simple Queue Service

SDB=Amazon SimpleDB

EBS=Elastic Block Storage

BS=Billing Service

In (7.2), C_{EC2}, C_{S3}, C_{SQS}, C_{SDB} are Amazon Web Services (AWS) consumers. (7.3) includes major components in ACA infrastructure:

WEB= Web server infrastructure

RT=Network Router

ELB=Elastic Load Balancer

AVM=Virtual EC2 Instance

ASP=Amazon Service Provider (AppServer)

VPN=Virtual Private Network

DS=Data Storage

(7.4) contains main components of ACA service management:

CWM=Cloud Watch Management

RM=Resource Management

BM=Billing Management

(7.5) is Amazon SOA Process:

KVS=Key Value Store [57]

PWF=Service Provisioning Workflow

(7.6) includes ACA deployment model and service delivery model.

Therefore, we have

$$CEA_{ACA}=\langle S_{ACA}, C_{ACA}, D_{ACA}, SI_{ACA}, SM_{ACA}, SP_{ACA}, SQ_{ACA}, SD_{ACA}\rangle, \tag{7.7}$$

And Table 7.11 shows that ACA is an instance of ECSA.

Table 7.11. Evaluating Amazon Cloud Architecture

CEA Evaluation Form		
Condition	Satisfy	Rationale
(1)	Yes	Service oriented and cloud enabled. It can be described by the (5.2)-(5.4). Please see from (7.1) to (7.7).
(2)	Yes	ACA is an EPUC.
(3)	Yes	It satisfies cloud quality attributes based on analysis in section 5.3.8. • ACA achieves dynamic scalability through Auto Scaling which allows you to automatically scale your Amazon EC2 capacity up or down according to conditions you define. With Auto Scaling, you can ensure that the number of Amazon EC2 instances you're using scales up seamlessly during demand spikes to maintain performance, and scales down automatically during demand lulls to minimize costs. • ACA also supports multiple location resilience for failure, so it keeps up to 99.95% high availability for each EC2 Region.. • ACA provides several security mechanisms for securing users' resources [12]. • ACA has high reliability through SLM. • ACA interoperability through supporting multiple protocol messaging interfaces – SOAP and REST. • ACA improves its performance through adopting high performance computing (HPC) clusters. • ACA is of higher flexibility through provisioning multiple types of EC2 instance and multiple options on CPU, OS, and Storage resources. • ACA provides mechanisms for higher accountability through its CWS, SLM and Billing Service (BM).

Although Amazon cloud satisfies the conditions as an instance of ECSA style, it has its weakness in its architecture design. The 12 hours-long outage [20] on 04/21/2011 of Amazon's IaaS brought many services and websites down. The outage stemmed from a human error in configuration change for its network upgrade in Amazon's big data center in Northern Virginia. Our cloud quality ontology can provide a good reasoning as to why the disaster outage happened in the Amazon cloud system. First of all, the reliability of the Amazon cloud architecture needs to be improved. The problem began early on Thursday morning and continued into Friday 04/22/2011. The DR duration is longer than 12 hours, so Amazon cloud reliability and DR capacity is relative low. Amazon cloud availability is based on its "availability zones" (see Figure 7.11). However, the availability zones only support failover within the same datacenter. If a disaster happens, such as the datacenter failure on 04/21/2011, there is no mechanism for datacenter failover which reduces availability. Moreover, the Amazon cloud service lacks high visibility to its service consumers. Therefore, consumers have no idea regarding what was happening. The lessons learned from the Amazon public cloud outage are (1) Cloud also has a single point failure if the failure control and recoverability is not addressed well by design; (2) design tradeoffs of cloud architecture among quality attributes are very important. Supporting failover and DR across datacenters is more expensive than doing that just across availability zones inside a single datacenter. However, we have to consider multiple ways to handle different failures to guarantee high availability. (3) One of the goals of cloud computing is sharing resources. However, cloud consumers and cloud providers need to share risks as well. This means that they both need to improve their applications to reach higher security and availability. Some customers of Amazon cloud, such as Zynga and Netflix, were not impacted by the

Amazon cloud outage, since they have better architecture design for failure. However, cloud providers should increase their architecture transparency and system visibility, so that their consumers can make their DR plan and improve their design for dealing with failures.

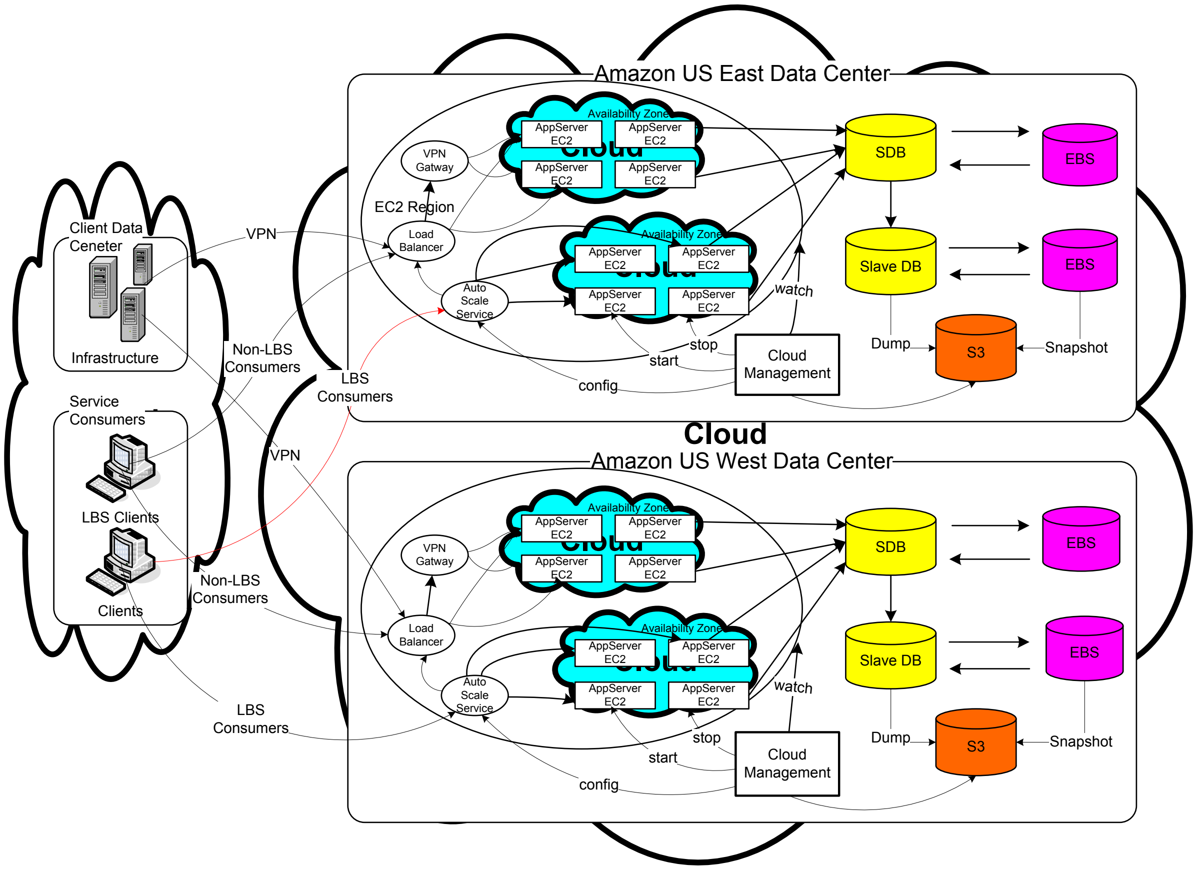


Figure 7.11. Amazon Cloud Architecture

7.4 Case Studies of ECSA-SLA

This section discusses two use cases of ECSA-SLA: SLA-Aware Private Cloud Enterprise Architecture and SLA-Aware Public Cloud Enterprise Architecture.

7.4.1 SLA-Aware Private Cloud Architecture

Figure 7.12 depicts an SLA-Aware private cloud enterprise architecture. In this case, business services and applications are running in enterprise owned data center(s). Resources and SLA are managed by enterprise IT. The SLA-Awareness is implemented in policy-based SLM. Normally, automated dynamic negotiation between service consumers and service providers are not required.

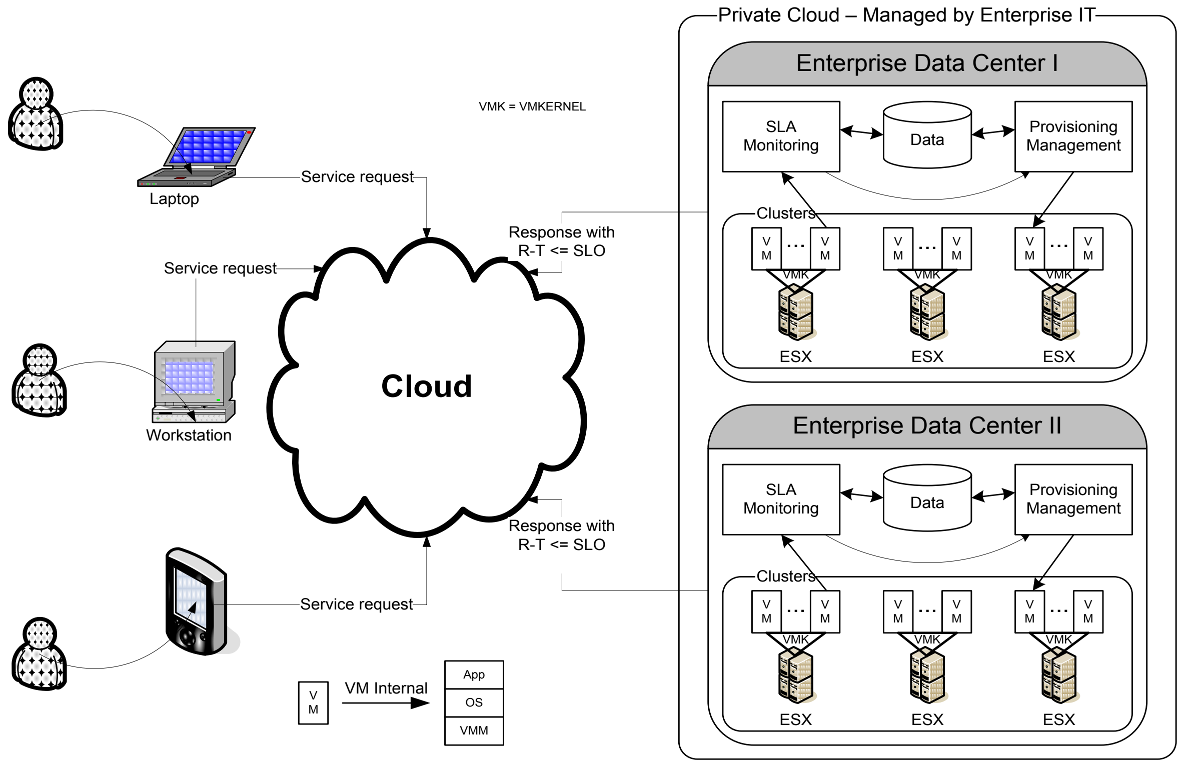


Figure 7.12. SLA-Aware Private Cloud Enterprise Architecture

7.4.2 SLA-Aware Public Cloud Enterprise Architecture

In public cloud enterprise architecture, service consumer (SC) and service provider (SP) are in different organizations. The SCs connect to the data center of SPs through the Internet. Therefore, security, availability, reliability and performance become concerns for SCs. It is very important to understand the amount that SCs are paying for the quality of services SCs are receiving. Any discrimination should be immediately attended to. To satisfy SCs' QoS requirements becomes a big challenge to design pubic cloud enterprise architecture. Figure 7.13

describes ideal SLA-Aware public cloud service enterprise architecture, such as the architecture proposed by SOA@SOI [190].

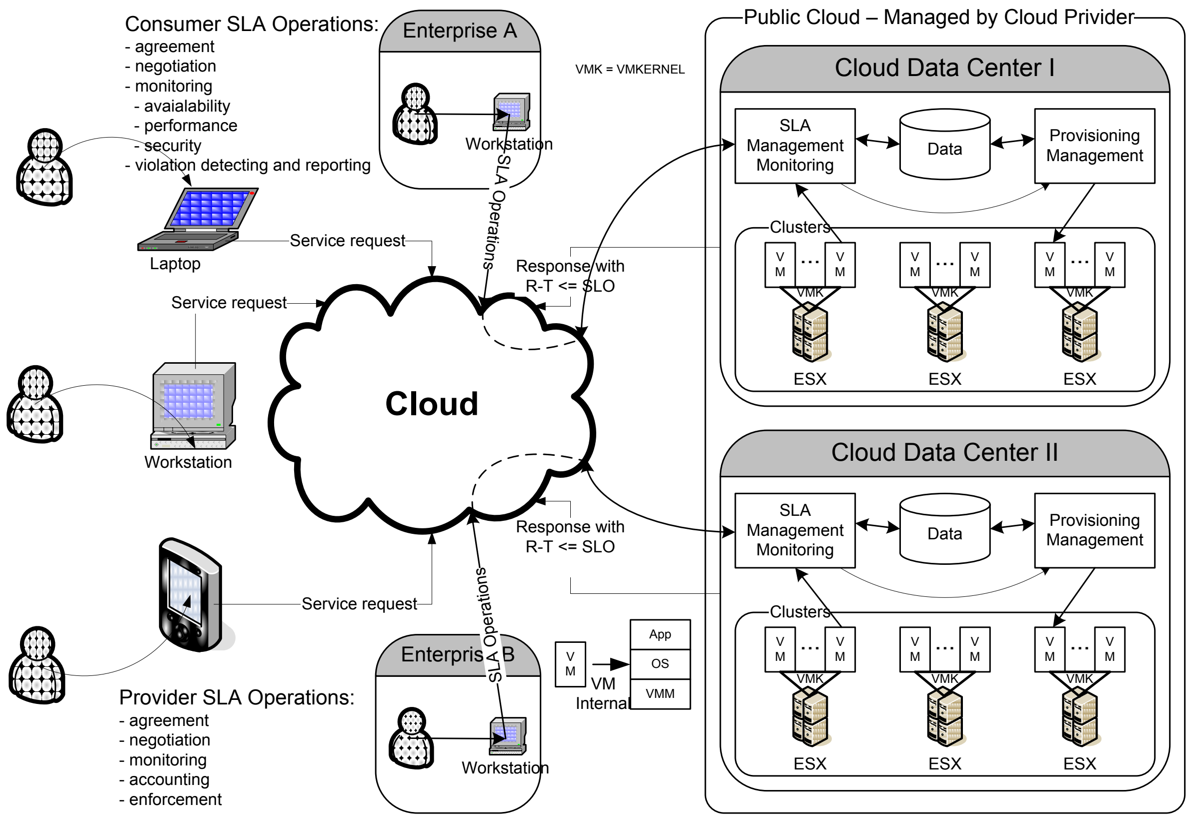


Figure 7.13. SLA-Aware Public Cloud Enterprise Architecture

As we defined ECSA-SLA in Chapter 6, the architectural style requires systems to:

- Adopt dynamic SLA and SLM;
- Consider SLA management and QoS is the first-class in systems;
- Provision of service based on SLA dynamically.

It is difficult to reach the level of SLA-Awareness in practice, since it requires systems with higher automation, adaptation and real time (RT) or close to RT monitoring system. It is easy to show that the main public cloud architectures, such as Amazon cloud, Google cloud and

Microsoft cloud are not SLA-Aware cloud service architecture. They all provide SLA for their service consumers, such as Amazon EC2 SLA [9]. However, the SLA is a static agreement document and, therefore, their system lacks dynamic SLM. Their systems reach a degree of SLA-Awareness, such as (1) semi-automated support service credit for SLA violation in their accounting system; for example, the Amazon EC2 service level is calculated on an annual basis, whereas the Amazon S3 service level is calculated over a monthly interval; (2) supporting close RT monitoring as well as automatically and dynamically service provisioning. Their system design is not directly based on SLA, but indirectly based on SLA, since security, availability and performance are taken into consideration at system design; otherwise, they will have a lot of SLA violations. Therefore, one can say that current main public cloud service architectures are SLA-based cloud service enterprise architectures. Let us define the maturity of SLA-Aware ECSA architecture as:

- Level 0 – Public Cloud Service Architecture in which SLA is not a constraint. It is not SLA-Aware, such as iCloud.
- Level 1 - Public Cloud Service Architecture based on SLA, in which SLA is a design constraint, but SLA is not considered as a first class concern in system. It is indirectly SLA-Aware, such as Amazon cloud.
- Level 2 - Public Cloud Service Architecture driven by SLA, in which SLA is considered as one of the most important design constraints and the system supports semi-auto and semi-dynamic SLM. It is partially SLA-Aware, such as SLA-driven resource provisioning.
- Level 3 – ECSA-SLA style architecture which is fully SLA-Aware (please see Section 6.3), such as SOA@SOI architecture.

7.5 Summary

This chapter analyzes consistency and extension of enterprise architectural styles – ESOA and ECSA, and defines the instances of ESOA and ECSA and describes and evaluates several case studies based on architectural styles we defined in Chapters 4 to 6. The lessons learned from these case studies, such as experiences from Zynga and outage lessons from Amazon, are also discussed. The case studies and their lessons learned are very helpful in guiding the design of service-oriented and cloud-enabled enterprise architecture.

CHAPTER 8

CONCLUSIONS

"The soul of an architecture is found in its mechanisms that cut across the components of the system, thus yielding its essential structures and behaviors."

- Grady Booch

"An architectural style, then, defines a family of such systems in terms of a pattern of structural organization. More specifically, an architectural style defines a vocabulary of components and connectors, and a set of constraints on how they can be combined."

- Mary Shaw and David Garlan

This chapter concludes the dissertation and discusses some future work.

8.1 Conclusions

Research of enterprise architectural styles has become more important to building high quality assurance and cost-effective enterprise systems. Specifically, research on service-oriented, cloud enabled enterprise architecture as well as architectural styles is gaining greater attention. From the beginning of the dissertation, I described the requirements and complexity of enterprise architectures and elaborated on the importance of modeling and analyzing service-oriented and cloud-enabled enterprise architectural styles. In this dissertation, several major service-oriented enterprise architectural styles ESOA and cloud enabled service-oriented enterprise architectural styles ECSA are described and analyzed based on the proposed framework [204] and ontology-based modeling methodology [168].

Both ESOA and ECSA serve as an integrated set of components (service consumers, services, processes, infrastructures and management) which outline the structure of service-oriented enterprise architectures.They also serve as a coordinated set of constraints that attempt to guide the building of service-oriented service systems to achieve enterprise non-functional requirements, such as security, performance and availability.

The following contributions to the field of Information and Computer Science and Software Engineering are included in this dissertation:

- A framework for modeling and analyzing service-oriented and cloud-enabled enterprise architectural styles. The framework is based on ontology-based modeling technology. It gives an insight into the design principles, structures, and behaviors for both functional and non-functional aspects of ECSA architectures and provides further understanding of the complex service-oriented and cloud-enabled enterprise architecture through the architectural styles ESOA and ECSA.
- An ECSA quality attributes tradeoff ontology which provides design principles for achieving high quality in ECSA style systems.
- A classifications of service-oriented and cloud enabled enterprise architectural styles by their structural characteristics and architectural properties. It can be applied to architecture decision making and to enterprise application system design referencing.
- A novel enterprise architectural style, ECSA, which combines both ESOA nad Cloud Computing. It can be used for understanding the current wave of enterprise architecture

movement and applied for designing SOA-centric and cloud enabled enterprise systems with higher quality.

- A formal analysis of enterprise architectural style consistency, extension and instantiation.
- A set of case studies for ESOA as well as ECSA style enterprise systems, which includes their descriptions, analysis and evaluation. Lessons learned from all these case studies are helpful for building better enterprise architecture practices.

Many enterprise architectures are instances of ESOA or ECSA. Although concrete enterprise architectures vary, they share some common components, connectors and constraints and follow common design principles. Specifically, consisting of service consumers, service processes, infrastructure, and management, service-oriented enterprise architectures are constrained by a set of quality attributes and need to follow common SOA design principles. For cloud-enabled service-oriented enterprise architectures, cloud specific components are added into ESOA style systems, such as IaaS, PaaS and SaaS. They have more specific constraints, such as elastic scalability and need to follow particular cloud computing principles, such as dynamic infrastructure as well as service provisioning, and virtualization. Enterprise architectural styles emphasize all common components, connectors, constraints, their relationship, and design principles. Not all instances are 100% in matching the certain style's properties and quality attributes. We have introduced a concept of maturity of an instance (an concrete enterprise architecture) of certain styles in Section 7.4. As a method of classification, each concrete enterprise architecture can be considered as an instance of an architectural style, such as two instances of ECSA-SLA: Amazon cloud and Google cloud. There are some differences between them – the Amazon cloud is more mature than the Google cloud, whose SLA is still in beta stage. Building standard and evaluation methodology for evaluating style instance's maturity is one of my future research interests.

In an ideal Information Technology (IT) world, the implementation of service-oriented and/or cloud-enabled enterprise architectures should match their design based on architectural styles. However, in the real IT world, some of the implementation fails to match the SOA and/or cloud design principles because of legacy experience or improper constraints tradeoff. This dissertation proposed a framework and defined ESOA and ECSA styles that are helpful for evaluating some of the broken links between implementation and design.

8.2 Future Work

The service-oriented enterprise architecture is a relevant new distributed computing paradigm, and cloud computing is an even newer distributed computing model. Their theory, standard and practices are not mature. However, they bring us large opportunities for future research. The future work includes, but is not limited to the following aspects:

- Developing more rigorous methods for analyzing and evaluating ESOA and ECSA enterprise architectural styles;
- Developing concrete methods for analyzing and evaluating style instance maturity;
- Application of the ontology-based framework.
- More in-depth tradeoff analysis of ESOA and ECSA architectural quality attributes.

Studies are never ending. Future work needs to focus more on practical applications of proposed framework of enterprise architectural styles and extend the framework to broader areas, such as enterprise wireless service computing.

REFERENCES

[1]. van der Aalst, W. M. P. and A. H. M. ter Hofstede, YAWL: yet another workflow language, Information Systems, Vol. 30, No. 4, 2005, pp. 245-275.

[2]. Abowd, G., R. Allen and D. Garlan, "Formalizing Style to Understand Descriptions of Software Architecture", ACM Trans. Software Eng. And Methodology, vol. 4, no. 4, 1995, pp.319-364.

[3]. Agopyan, A., H. Huebler, T. Puah, T. Schalze, D.S. Vilageliu and M. Keen, WebSphere Application Server V7: Concept, Planning, and Design, IBM Redbook, Feb. 2009

[4]. Aier, S., Maximilian Ahrens, Matthias Stutz and Udo Bub, Deriving SOA Evaluation Metrics in an Enterprise Architecture Context, Lecture Notes in Computer Science, Volume 4907/2009, pp. 224-233.

[5]. Alberti, M., F. Chesani, M. Gavanelli, E. Lamma, P. Mello, M. Montali, S. Storari, and P. Torroni, Computational Logic for Run-time Verification of Web Services Choreographies: Exploiting the SOCS-SI Tool, WS-FM 2006, LNCS 4184, pp. 58-72, 2006.

[6]. Allen, R., Formalism and Informalism in Software Architectural Style: a Case Study, Proceedings of the First International Workshop on Architectures for Software System, pp. 1-8, 1995

[7]. Allen, R. and D. Garlan, A formal basis for architectural connection, ACM Transactions on Software Engineering and Methodology 6(3), pp. 213-249, 1997.

[8]. Amazon Web Services, http://aws.amazon.com/about-aws/

[9]. Amazon, EC2 SLA, http://aws.amazon.com/ec2/

[10]. Amazon, Auto Scaling and load banlance, http://aws.amazon.com/autoscaling/

[11]. Amazon.com, Instance Metadata, http://docs.amazonwebservices.com/AWSEC2/latest/DeveloperGuide/index.html?AESDG-chapter-instancedata.html

[12]. Amazon, Amazon Web Services: Overview of Security Processes, 2009. http://awsmedia.s3.amazonaws.com/pdf/ AWS_Security_Whitepaper.pdf

[13]. Amrhein, D., Bringing Cloud Computing to SOA, http://websphere.sys-con.com/node/981796

[14]. Andersson, J. and P. Johnson, "Architectural integration styles for large-scale enterprise software systems", Enterprise Distributed Object Computing Conference, 2001. EDOC '01. Proceedings. Fifth IEEE International, Sept. 4-7, Seattle, USA, 2001, pp.224-236

[15]. Andrieux, A., K. Czajkowski, A. Dan, H. Ludwig, T. Nakata, J. Pruyne, J. Rofrano, S. Tuecke and M. Xu, *Web Service Agreement Specification (WS-Agreement)*. Retrieved from http://www.ogf.org/documents/GFD.107.pdf, 2007

[16]. Anstett, T., F. Leymann, R. Mietzner and S. Strauch, Towards BPEL in the Cloud: Exploiting Different Delivery Models for the Execution of Business Processes, IEEE 2009 Congress on Services, p.670-685

[17]. Apache ODE RESTful BPEL, http://ode.apache.org/restful-bpel-part-i.html

[18]. Arora, S., Business Process Management, Process is the Enterprise, LuLu Publish, May. 17, 2005

[19]. Baader, F., D. McCuiness, D. Nardi and P.P. Schneider (Eds.), The Description Logic Handbook, Cambridge University Press, 2003

[20]. Babcock, C., Lessons From FarmVille, p. 29-57, InformationWeek, Issue 1300, May 16, 2011

[21]. Badidi, E., L. Esmahi, M. Adel Serhani and M. Elkoutbi, WS-QoSM: A Broker-based Architecture for Web Services QoS Management. *Innovations in Information Technology* , pp. 1-5, 2006.

[22]. Bahree, A., S. Cicoria, D. Mulder, N. Pathak and C. Peiris, Pro WCF: Practical Microsoft SOA Implementation, Apress, 2007

[23]. Baresi, L., R. Heclel, S. Thone and D. Varro, "Modeling and Validation of Service-Oriented Architectures Application vs. Style", ESEC/FSE'03, Sept. 1-5, 2003.

[24]. Bass, L., P. Clements and R. Kazman, Software Architecture in Practice, Addison-Wesley, 2003

[25]. Bernardo, M., P. Ciancarini and L. Donatirllo, Architecting Families of Software Systems with Process Algebras, ACM Trans. on Software Engineering and Methodology Vol. 11 pp. 386 - 426, October 2000.

[26]. Bhoj, P., S. Singhal and S. Chutani, SLA Management in federated environments, *Computer Networks,* Vol. 35, pp. 5-24, 2001.

[27]. Bianco, P., R. Kotermanski and P. Merson, "Evaluating a Service-Oriented Architecture", Technical Report CMU/SEI-2007-TR-015

[28]. BitCurrent, New report on cloud Computing Performance, 2010, http://www.bitcurrent.com/new-report-on-cloud-performance/#

[29]. Boreale, M., R. Bruni, L. Caires, R. De Nicola, I. Lanese, M. Loreti, F. Martins, U. Montanari, A. Ravara, D. Sangiorgi, V. Vasconcelos and G. Zavattaro, SCC: A Service Centered Calculus, WS-FM 2006, LNCS 4184, pp.38-57,2006.

[30]. Brandic, I., S. Venugopal, M. Mattess and R. Buyya, Towards a Meta-Negotiation Architecture for SLA-Aware Grid Services. Technical Report GRIDS-TR-2008-10, 2008.

[31]. Brodkin, J., Gartner: Seven cloud-computing security risks Data integrity, recovery, privacy and regulatory compliance are key issues to consider, Network World, 2009 at http://www.networkworld.com/news/2008/070208-cloud.html

[32]. Broy, M., I.H. Kruger and M. Meisinger, "A Formal Model of Services", ACM Transaction Software Eng. And Methodology, vol.16, no. 1, 2007.

[33]. Burke, B. and R. Monson-Haefel, Enterprise JavaBeans 3.0, O'Reilly, 2006

[34]. Butler, J., "Creating a UML Profile from the CBDI SAE Meta Model", CBDI Journal, Jan. 2008.

[35]. Buyya, R., C. S. Yeo, S. Venugopal, J. Broberg and I. Brandic, Cloud computing and emerging IT platforms: Vision, hype, and reality for delivering computing as the 5th utility, *Future Gener. Comput. Syst.*, Vol. 25, No. 6. (2009), pp. 599-616.

[36]. Cakic, J. and R. F. Paige, Origins of the Grid Architectural Style, Proceedings of the 11th IEEE International Conference on Engineering of Complex Computer Systems, 2006.

[37]. Cesare, P., Z. Olaf; L. Frank, "RESTful Web Services vs. Big Web Services: Making the Right Architectural Decision", *17th International World Wide Web Conference (WWW),* Beijing, China, 2008.

[38]. Chappell, D.A., Enterprise Service Bus, O'Reilly, 2004.

[39]. Chappell, D. and D. Berry, Next-Generation Grid-Enabled SOA: Not Your MOM's Bus, SOA Magazine, Issue XIV, 2008

[40]. Chappell, D. and D. Berry, SOA – Ready for Primetime: The Next-Generation, Grid-Enabled Service-Oriented Architecture, SOA Maganzing, Sept. 2007

[41]. Chen, Y. and W.T. Tsai, Distributed Service-Oriented Software Development, Kendall Hunt Pub Co, 2008

[42]. Choi, Si Won, Jin Sun Her and Soo Dong Kim, "Modeling QoS Attributes and Metrics for Evaluating Services in SOA Considering Consumers' Perspective as the First Class Requirement", in: Asia-Pacific Service Computing Conference, The 2nd IEEE, 11-14 Dec. 2007, pp. 398-405

[43]. Chothia, T. and J. Kleijn, Q-Automata: Modeling the Resource Usage of Concurrent Components. *Electronic Notes in Theoretical Computer Science*, Vol. 175, pp. 153-167, 2007.

[44]. Chung, Jen-Yao, K.J. Lin, Richard G. Mathieu, "Web Services Computing - Advancing Software Interoperability," IEEE Computer, 35-37, October 2003

[45]. Chung, L. and N. Subramanian, Adaptive System/Software Architecture. Journal of Systems Architecture, 2003.

[46]. Chung, L., B.A. Nixon, E. Yu and J. Mylopoulos, *Non-functional requirements in software engineering*, Springer, 2000.

[47]. Ciancarini, P. and C. Mascolo, Analyzing and Refining an Architectural Style, Proceedings of the 10th International Conference of Z Users on The Z Formal Specification Notation, pp. 349-368, 1997

[48]. Clarkin, L. and J. Holmes, Enterprise Mashups, The Architecture Journal, 13 (2007)

[49]. Clements, P., F. Bachmann, L. Bass, D. Garlan, J. Ivers, R. Little, R. Nord and J. Stafford, Documenting Software Architectures Views and Beyond, Addison-Wesley, 2002

[50]. CIS, The CIS Security Metrics, 2009, https://www.cisecurity.org/tools2/metrics/CIS_Security_Metrics_v1.0.0.pdf

[51]. Comuzzi, M., W. Theilmann, G. Zacco, C. Rathfelder, C. Kotsokalis and U. Winkler, A Framework for Multi-level SLA Management. The eighth International Conference on Service Oriented Computing (ICSOC), 2009.

[52]. Creeger, M., Cloud Computing: An Overview, Acmqueue, 2009

[53]. Curbera, F. and N. Mukhi, "Metadata-driven middleware for Web Services", WISE 2003, Proceedings of the Fourth International Conference, 10-12 Dec. 2003 pp. 278 – 283

[54]. Curbera, F., M. Duftler, R. Khalaf and D. Lovell, Bite: Wrokflow Composition for the Web, International Conference on Services Oriented Computing (2007), LNCS 4749, pp. 94-106, 2007

[55]. Dan, A. H. Ludwig, and G. Pacifici (2003). *Web Services Differentiation with Service Level Agreement*. Retrieved from http://www.ibm.com/developerworks/library/ws-slafram/

[56]. Davies, J., SOA: BEA, Apress, 2007

[57]. DeCandia, G., D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pichin, S. Sivasubramanian and P. Vosshall, Dynamo: Amazon's Highly Available Key-value Store, SOSP'07, Oct. 14-17, USA

[58]. Decker, G., F. Puhlmann and M. Weske, "Formalizing Service Interactions", LNCS 4102, pp. 414-419, 2006.

[59]. Dillon, T. S., Chen Wu and Elizabeth Chang, Reference Architectural Styles for Service-Oriented Computing, LNCS 4672, pp. 543-555, 2008.

[60]. DMTF, Open Virtualization Format Specification, 2009, http://www.dmtf.org/standards/published_documents/DSP0243_1.0.0.pdf

[61]. Dobson, G. and A. Sanchez-Macian, Towards unified QoS/SLA Ontologies. *Proceedings of the IEEE Services Computing Workshops*, pp. 169-174, 2006.

[62]. Dong, J., R. Paul, and L.-J. Zhang, High Assurance Service-Oriented Architecture, IEEE Computer, Volume 41, Issue 8, Pages 22-23, August 2008.

[63]. Dong, J., R. Paul, and L.-J. Zhang, High Assurance Services Computing, Springer, 2009

[64]. Dornemann, T., E. Juhnke and B. Freisleben, On-Demand Resource Provisioning for BPEL Workflow Using Amazon's Elastic Compute Cloud, The Proceddings of 9th IEEE/ACM International Symposium on Cloud Computing and the Grid, p. 140-147, 2009.

[65]. Erasala, N., David C. Yen and T. M. Rajkumar, Enterprise Application Integration in the electronic commerce world, Computer Standards & Interfaces, Vol. 25, Issue 2, May 2003, Pages 69-82

[66]. Erl, T., Service-Oriented Architecture, Pearson Education, 2005

[67]. Erl, T., SOA Principles of Service Design, Prentice Hall, 2008

[68]. Erl, T., SOA Design Patterns, Prentice Hall, 2008

[69]. Ferguson, D. F., Dennis Pilarinos and John Shewchuk, The Internet Service Bus, The Architecture Journal, 13 (2007)

[70]. Fielding, R. T., Architectural Styles and the Design of Network-based Software Architectures, PhD Thesis, University of California, Irvine, 2000.

[71]. Fielding, R. T. and R. N. Taylor (2002-05), "Principled Design of the Modern Web Architecture", *ACM Transactions on Internet Technology (TOIT)* (New York: Association for Computing Machinery) **2** (2): 115–150

[72]. Foster, Ian, Yong Zhao, Ioan Raicu and Shiyong Lu, Cloud Computing and Grid Computing 360-Degree Compared, *Grid Computing Environments Workshop, 2008. GCE '08* In Grid Computing Environments Workshop, 2008. pp. 1-10.

[73]. Fu, Y., Z. Dong, X. He, An approach to web services oriented modeling and validation, Proceedings of the 2006 international workshop on Service-oriented software engineering, pp. 81 – 87, 2006.

[74]. Gall, Nick, Why WOA vs. SOA Doesn't Matter? http://www.itbusinessedge.com/item/?ci=47620&sr=1, 2008

[75]. Gamble, M. T. and R. Gamble, Monoliths to Mashup: Increasing Opportunistic Assets, 25(6):71-79, 2008 IEEE Software

[76]. Gao, T., H. Ma, I.-L. Yen, F. Bastani and W.-T. Tsai, Toward QoS Analysis of Adaptive Service-Oriented Architecture. *IEEE International Symposium on Service-Oriented System Engineering (SOSE*) , pp. 219-226, 2005.

[77]. Garlan, D., R.T. Monroe and D. Wile, Acme: Architectural Description of Component-Based Systems, Book: Foundations of component-based systems, Cambridge University Press New York, NY, USA, pp. 47-67, 2000

[78]. Garlan, D., Foraml Modeling and Analysis of Software Architecture: Components, Connectors, and Events, LNCS 2804, pp. 1-24, 2003

[79]. Garlan, D. and B. Schmerl, Architecture-driven modeling and analysis, in Tony Cant (Ed.), Proceedings of the 11th Australian Workshop on Safety Related Programmable Systems (SCS'06), Conferences in Research and Practice in Information Technology, Vol. 69, 2006

[80]. Gibbens, R., R. Mason and R. Steinberg, Internet service classes under competition. *IEEE Journal on Selected Areas in Communications*, Vol. 18, No. 12, pp. 2490-2498, 2000.

[81]. Giesecke, S., "Middleware-induced styles for enterprise application integration" In Software Maintenance and Reengineering, 2006. CSMR 2006. Proceedings of the 10th European Conference on 2006.

[82]. Gorton, I. and A. Liu, An Architects' Guide to enterprise application integration with J2EE and .NET, Proceedings of the International Conference on Software Engineering, ICSE 2005, pp. 726-727

[83]. Governor, J., D. Hinchcliffe and D. Nickull, Web 2.0 Architectures, Oreilly Media, May 2009

[84]. Guidi, C., R. Lucchi "Formalizing mobility in Service Oriented Computing", Journal of Software, Vol. 2, No. 1, 2007

[85]. Harvard Research Group, Availability Environment Classification, Publication # 5968-6578EUC. http://www.hrgresearch.com/pdf/MarathonHPPAgpl060499d.pdf

[86]. Harvard Research Group, Cloud Computing – Introduction, http://www.hrgresearch.com/Cloud%20Computing.html

[87]. Hasselmeyer, P., C. Qu, L. Schubert, B. Koller and P. Wieder, *Towards Autonomous Brokered SLA Negotiation, from "Exploiting the Knowledge Economy: Issues, Applications, Case Studies",* IOS Press, Amsterdam.

[88]. Heisel, M. and N. Levy, Using LOTOS Patterns to Characterize Architectural Styles, LNCS 1214, pp. 818-832, 1997

[89]. Heiser, J. and Mark Nicolett, Assessing the Security Risks of Cloud Computing, 3 June 2008, http://www.gartner.com/DisplayDocument?id=685308

[90]. Henson, M. D., SOA Using Java Web Services, Prentice Hall, 2007

[91]. High, R., S. Kinder and S. Graham, "IBM's SOA Foundation: An Architectural Introduction and Overview", http://www.ibm.com/developerworks/webservices/

[92]. Hinchcliffe, D., The SOA with reach: Web-Oriented Architecture, 2006 at http://blogs.zdnet.com/Hinchcliffe/?p=27

[93]. Hirsch, D., P. Inverardi and U. Montanari, Modeling software architectures and styles with graph grammar and constraint solving, in First Working IFIP Conference on Software Architecture, Feb., 1999

[94]. Hohpe, G., and Bobby Woolf, Enterprise Integration Patters: Designing, Building, and Deploying Messaging Solutions, Addison-Wesley, ISBN 0321200683, 2002

[95]. HP, The Cloud and SOA, http://www.hp.com/hpinfo/analystrelations/wp_cloudcomputing_soa_capgemini_hp.pdf

[96]. Huhns, M.N. and M.P. Singh, Service-oriented computing: key concepts and principles, IEEE Internet Computing, Vol. 9, N. 1, 2005

[97]. IBM sMash, http://www.ibm.com/developerworks/ibm/library/i-zero1/

[98]. IBM, Seeding the Clouds: Key Infrastructure Elements for Cloud Computing, Feb. 2009. http://www-935.ibm.com/services/uk/cio/pdf/oiw03022usen.pdf

[99]. Josuttis Braunschweig, N. M., SOA in Practices, O'Reilly Media, Inc, 2008

[100]. Kacem, M.H., M.Jmaiel, A.H. Kacem and K. Drira, Using UML2.0 and GG for describing the dynamic of software architectures, Information Technology and Applications, 2005. ICITA 2005. Third International Conference on 4-7 July 2005

[101]. Karasavvas, K., M. Antonioletti, M. Atkinson, N. C. Hong, T. Sugden, A. Hume, M. Jackson, A. Krause and Charaka Palansuriya, Introduction to OGSA-DAI Services, LNCS Vol. 3458, 2005, pp. 1-12

[102]. Kavantzas, N., D. Burdett, G. Ritzinger, T. Fletcher, Y. Lafon, and C. Barreto, Web Services Choreography Description Language Version 1.0, 2003.

[103]. Kavantzas, N., http://lists.w3.org/Archives/Public/www-archive/2004Jun/att-0008/WS-CDL-April2004.pdf, 2004

[104]. Keen, M., J. Bond, J. Denman, S. Foster, S. Husek, B. Thompson and H. Wylie, Patterns: Integrating Enterprise Service Buses in a Service-Oriented Architecture, IBM Redbooks, 2005.

[105]. Keller, A. and H. Ludwig, The WSLA Framework: Specifying and Monitoring Service Level Agreement for Web Service, *Journal of Network and System Management* , Vol. 11, No. 1, pp. 57-81, 2003.

[106]. Kephart, J.O. and D. Chess, The Vision of Autonomic Computing. *IEEE Computer* (V. 36, N. 1, pp. 41-50, 2010.

[107]. Khare, R. and R. Taylor, Extending the Representational State Transfer (REST) Architectural Style for Decentralized Systems, Proceedings of the 26th International Conference on Software Engineering, 428-437, 2004

[108]. Kim, J.S. and D. Garlan, Analyzing Architectural Styles with Alloy, In Proc. Workshop on the Role of Software Architecture for Testing and Analysis, 2006

[109]. Klein, J., Architecture for HIPAA Compliance, Gartner Symposium ITxpo 2001 (Gartner Research)

[110]. Klein, M. H., R. Kazman, L. Bass, J. Carriere, M. Barbacci, and H. Lipson, Attribute-Based Architecture Styles, IFIP Conference Proceedings; Vol. 140, 1999, 225-244

[111]. Kopecky, J., T. Vitvar, C. Bournez and J. Farrell, SAWSDL: Semantic Annotations for WSDL and XML Schema, IEEE Internet Computing, Vol. 11, Issue 6, 2007

[112]. Krafzig, D., K. Banke and D. Slama, Enterprise SOA: Service-Oriented Architecture Best Practices. Prentice Hall, Englewood Cliffs (2004)

[113]. Kritikos, K. and D. Plexousakis, *QoS-Based Web Service Description and Discovery*, from http://ercim-news.ercim.eu/qos-based-web-service-description-and-discovery, 2008.

[114]. Kruchten, P., H. Obbink and J. Stafford, “The Past, Present, and Future of Software Architecture”, IEEE Software, vol. 23, no. 2, 2006

[115]. Lakshman, G. and P. Manish, How the Cloud Stretches the SOA Scope, p. 36-41, V. 21, Architecture Journal 2009

[116]. Laliwala, Z. and S. Chaudhary, Event-Driven Service-Oriented Architecture, Service Systems and Service Management, 2008, pp. 1-6.

[117]. Lara, R., M. Stollberg, A. Polleres, C. Feier, C. Bussler and D. Fensel, Web Service Modeling Ontology, Applied Ontology, 1(1), pp. 77-106, 2005

[118]. Lankhorst, M., ArchiMate: A Service-Oriented Enterprise Architecture Modeling Language, OMG Technical Meeting, SOA WG 12/06/2005.

[119]. Lawson (blog), L., Identifying the Synergy Between SOA and the Cloud *Mar 19, 2009.* http://www.itbusinessedge.com/cm/blogs/lawson/identifying-the-synergy-between-soa-and-the-cloud/?cs=31219

[120]. Leavitt, N., "Is Cloud Computing Really Ready for Prime Time?," Computer, vol. 42, no. 1, pp. 15-20, Jan. 2009, doi:10.1109/MC.2009.20

[121]. Lee, J., et al., “Integrating Service Composition Flows with User Interactions”, Proc. of IEEE Service-Oriented System Engineering, 2008, pp. 103-108.

[122]. Leymann, F., Choreography for the Grid: towards fitting BPEL to the resource framework, Concurrency and Computation: Practice & Experience, Vol. 18, No. 10, 2006, pp. 1201-1217.

[123]. Liang, D., “Servicetizing User Experiences for Complex Business Application”, Proc. of IEEE Service-Oriented System Engineering, 2006, pp. 147-155.

[124]. Lindsk, E., IBM Brings SOA to Cloud, http://it.tmcnet.com/topics/it/articles/55460-ibm-brings-soa-the-cloud.htm

[125]. Linthicum, D. S., Cloud Computing and SOA Convergence in Your Enterprise, Addison-Wesley, Sept. 2009.

[126]. Liu, Y., A.H. Ngu and L.Z. Zeng, QoS computation and policing in dynamic web service selection. *Proceedings of the 13th International World Wide Web conference on Alternate track papers & posters*, pp. 66-73, 2004.

[127]. Lublinsky, B., Defining SOA as an Architectural Style:Align your business model with technology, http://www.ibm.com/developerworks/architecture/library/ar-soastyle/, 2007

[128]. Ludwig, H., WS-Agreement Concepts and Use: Agreement-Based, *Service-Oriented Architecture, Service-Oriented Computing* , pp. 199-228, The MIT Press, 2009.

[129]. Ludwig, H., A. Keller, A. Dan and R. King, A Service Agreement Language for Dynamic Electronic Services. *Electronic Commerce Research* ,Vol. 3, pp. 43-59, 2003.

[130]. Lynch, N. and S. Gilbert, Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services”, *ACM SIGACT News*, Vol. 33 Issue 2 (2002), pp. 51-59

[131]. Mahbub, K. and Spanoudakis, Monitoring WS-Agreements: An Event Calculus–Based Approach, *Test and Analysis of Web Services, 2007.*

[132]. Martin, D., M. Burstein, D. McDermott, S. McIlraith, M. Paolucci, K. Sycara, D. L. McGuinness, E. Sirin and N. Srinivasan, Bringing Semantics to Web Services: The OWL-S Approach, LNCS Vol. 3387, pp. 26-42, 2005

[133]. Mazzara, M. and S. Govoni, A Case Study of Web Services Orchestration, LNCS 2005, pp. 1-16.

[134]. McGough, A.S., A. Akram, D. Colling, L. Guo, C. Kotsokalis, M. Krznaric, P. Kyberd and J. Martyniak, Enabling Scientists Through Workflow and Quality of Service, *Grid Enabled Remote Instrumentation*, pp. 345-359, Springer, 2009.

[135]. Mcgovern, J., O. Sims, A. Jain and M. Little, Enterprise Service Oriented Architecture, Springer, 2006.

[136]. Meng, S., QCCS: A Formal Model to Enforce QoS Requirements in Service Composition. *Proceedings of the First Joint IEEE/IFIP Symposium on Theoretical Aspects of Software Engineering,* pp. 389-400, 2007.

[137]. Mecella, M., F. P. Presicce and B. Pernici, "Modeling E-service Orchestration through Petri Nets", LNCS 2444, pp.38-47, 2002

[138]. Medvidovic, N., R.N. Taylor, A classification and comparison framework for software architecture description languages, Software Engineering, IEEE Transactions on Volume 26, Issue 1, Jan. 2000, pp. 70 – 93

[139]. Le Metayer, D., Software architecture styles as graph grammars, ACM SIGSOFT Software Engineering Notes, Vol. 21, No. 6, pp. 15-23, 1996

[140]. Michlmayr, A., F. Rosenberg, C. Platzer, M. Treiber and S. Dustdar, "Towards Recovering the Broken SOA Triangle – A Software Engineering Perspective", IW-SOSWE'07, Sept. 3, 2007.

[141]. Microsoft ESB Guidance for BizTalk Server 2006 R2, http://msdn.microsoft.com/en-us/libray/bb931189.aspx

[142]. Microsoft, Window Azure Platform, 2009, http://www.microsoft.com/**azure**

[143]. Milner, R., Communication and Mobile Systems: the -calculus, Cambridge University Press, 1999.

[144]. Misra, J. and WR Cook, Computation orchestration: A basis for wide-area computing, Journal on Software and System Modeling, pp. 1-26, 2006

[145]. Monroe, R.T., A. Kompanek, R. Melton and D. Garlan, Architectural Styles, Design Patterns, and Objects, IEEE Software, pp. 43-52, 1997

[146]. zur Muehlen, M., J. V. Nickerson and K. D. Swenson, Developing web services choreography standards – the case REST vs. SOAP, Decision Support Systems 40 (2005) 9-29

[147]. Mule Galaxy at http://mule.mulesource.org/display/MULE/Home

[148]. De Nicola, R., G. Ferrari, U. Montanari, R. Pugliese and E. Tuosto, A Formal Basis for Reasoning on Programmable QoS. *Lecture Notes in Computer Science*, Vol. 2772, pp. 436-479, 2003.

[149]. De Nicola, R., G. Ferrari, U. Montanari, R. Pugliese and E. Tuosto, A Process Calculus for QoS-Aware Applications. *Lecture Notes in Computer Science*, Vol. 3454, pp. 33-48, 2005.

[150]. Nurmela, T. and L. Kutvonen, Service Level Agreement Management in Federated Virtual Organizations. *Lecture Notes in Computer Science* (Vol. 4531, pp. 62-75, 2007.

[151]. OASIS MOWS V1.1 Specification http://www.oasis-open.org/committees/download.php/20574/wsdm-mows-1.1-spec-os-01.pdf

[152]. OASIS MUWS 1.1 Specification. http://www.oasis-open.org/committees/download.php/20576/wsdm-muws1-1.1-spec-os-01.pdf

[153]. OASIS SCA http://www.oasis-opencsa.org/sca

[154]. OASIS SOA Reference Model, http://www.oasis-open.org/

[155]. OASIS, Web Services Business Process Execution Language Version 2.0, OASIS, 2007, http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html

[156]. OASIS WSDM 1.1 Specification http://www.oasis-open.org/committees/tc_home.php?wg_abbrev=wsdm

[157]. OASIS, Web Service Resource Framework (WSRF) – Primer v1.2, 2006, http://docs.oasis-open.org/wsrf/wsrf-primer-1.2-primer-cd-02.pdf

[158]. O'Brien, L., L. Bass and P. Merson, "Quality Attributes and Service-Oriented Architectures", Technical Note, CMU/SEI-2005-TN-014.

[159]. O'Brien, L., L. Bass and P. Merson, "Quality Attributes and Service-Oriented Architectures", In Proceedings of the International Workshop on Systems Development in SOA Environments, 2007, pp. 3-7

[160]. OGF, "Open Grid Service Infrastructure (OGSI), http://www.ggf.org/documents/GFD.15.pdf, 2003

[161]. OMG, Service-Oriented Architecture Modeling Language, 2008-08-04, 2008, http://www.omgwiki.org/SoaML/doku.php?id=specification

[162]. *The Open Group, SLA Management Handbook*, ISBN: 1-931624-51-8, 2004**.**

[163]. Oracle Application Server 10g ESB http://www.oracle.com/technology/products/integration/esb/pdf/ds_esb_v10_1_2.pdf

[164]. Oracle, Architectural Strategies for Cloud Computing, 2009, http://www.oracle.com/technology/architect/entarch/pdf/architectural_strategies_for_cloud_computing.pdf

[165]. Ouzzani, M. and Athman Bouguettaya, Efficient Access To Web Services, IEEE Internet Computing, vol. 8, no. 2, 34-44, 2004.

[166]. Pahl, C. and Ronan Barrett, Layered Patterns in Modelling and Transformation of Service-Based Software Architectures, LNCS 4344, 2006, pp. 144-158.

[167]. Pahl, C., S. Giesecke and W. Hasselbring, An Ontology-Based Approach for Modeling Architectural Styles, LNCS 4758, pp. 60-75, 2007

[168]. Pahl, C., S. Giesecke and W. Hasselbring, Ontology-based modeling of architectural styles, Information and Software Technology, 51 (2009) 1739-1749

[169]. Peltz, C., "Web Services Orchestration and Choreography", IEEE Computer, vol. 36, no. 10, 2003, pp. 46-52.

[170]. Perry, D.E. and A.L. Wolf, "Foundations for the Study of Software Architecture," ACM Software Eng. Notes, vol.17, no. 4, 1992, pp. 40-52.

[171]. Piccinelli, G., A. Finkelstein, and S.L. Williams , Service-oriented workflow: the DySCo framework, Euromicro Conference, 2003. Proceedings. 29th, pp. 291-297, 2003.

[172]. Prescod, P., Roots of the REST/SOAP Debate, Extreme Markup Languages, (2002)

[173]. Reese, G., Cloud Application Architectures, O'Reilly Media, Inc, 2009

[174]. O'Reilly, T., What Is Web 2.0, (2005, Retrieved on 2006) O'Reilly Network.

[175]. Richardson, L. and Sam Ruby, "RESTful Web Services", O'Reilly, 2007

[176]. Rosenberg, D., Web-Oriented architecture and the rise of pragmatic SOA, blog (2008) at http://news.cnet.com/8301-13846_3-10031651-62.html

[177]. Rosenberg, F., F. J. Duftler, and R. Khalaf, Composing RESTful Services and Collaborative Workflows, 12(5):24-31,2008 IEEE Internet Computing

[178]. Schulte, W. and D. Sholler, SOA Overview and Guide to Research, Gartner Research Report (G00166742), 2009

[179]. ServiceMix, http://servicemix.apache.org

[180]. Shaw, M., Comparing architectural design styles, IEEE Software, 12 (6), 1995

[181]. Shaw, M. and D. Garlan, Formulations and Formalisms in Software Architecture, Lecture Notes in Computer Science, 1995, Volume 1000/1995, 307-323A

[182]. Shaw. M., Toward higher-level abstractions for software systems. *Data & Knowledge Engineering*, 5, 1990, pp. 119-128.

[183]. Shaw, M. and P. Clements, The Golden Age of Software Architecture, IEEE Software, pp. 31-39, 2006

[184]. Shaw, M. and D. Garlan, The Coming-of-Age of Software Architecture Research, Proceedings of the IEEE 23rd International Conference on Software Engineering, 2001

[185]. Shaw, M. and D. Garlan, Software Architecture, Prentice Hall, 1996

[186]. Schmid, M. and R. Kroeger, Decentralised QoS-Management in Service Oriented Architecture. *Lecture Notes in Computer Science*, Vol. 5053, pp. 44-57, 2008.

[187]. Singh, M. P., A.K. Chopra and N. Desai, Commitment-Based Service-Oriented Architecture, IEEE Computer, vol. 42, No. 11, 2009, pp 72-79

[188]. Singh, M. P. and M. N. Huhns, Service-Oriented Computing, John Wiley & Sons, 2005.

[189]. Skene, J., D.D. Lamanna, and W. Emmerich, Precise Service Level Agreement, Proceedings of the 26th International Conference on Software Engineering, 2004.

[190]. SLA@SOI, Empowering the service industry with SLA-aware infrastructures. Retrieved October 24, 2010, from http://sla-at-soi.eu/research/

[191]. Slomiski, A., On using BPEL extensibility to implement OGSI and WSRF Grid workflows, Concurrency and Computation: Practice and Experience, Vol. 18, No. 10, 1229-1241, 2005

[192]. Smith, R., Smart Web App Development, (2008) InformationWeek

[193]. Solanki, M., A. Cau, and H. Zedan, ASDL: A Wide Spectrum Language for Designing Web Services, ACM IW3C2, May 23-26, 2006.

[194]. Song, Y., Y. Li, H. Wang, Y. Zhang, B. Feng, H. Zang and Y. Sun, A Service-Oriented Priority-Based Resource Scheduling Scheme for Virtualized Utility Computing *Lecture Notes in Computer Science* , Vol. 5374, pp. 220-231, 2008.

[195]. SUN, Sun GlassFish Enterprise Service Bus: The Lightweight ESB, https://www.sun.com/offers/details/Lightweight_ESB.xml?cid=927706

[196]. SUN GlassFish, http://www.sun.com/software/products/glassfishv3_prelude/

[197]. SUN, Introduction to Cloud Computing Architecture, 2009, http://www.sun.com/featured-articles/CloudComputing.pdf

[198]. Sunbul, A., Abstract State Machines for the Composition of Architectural Styles, LNCS 1755, pp. 54-61, 2000

[199]. Szyperski, C. (2002). *Component Software: Beyond Object-Oriented Programming*. 2nd ed. Addison-Wesley Professional, Boston

[200]. Tang, L. and J. Dong, "A Survey of Formal Methods for Software Architecture", Proceedings of the International Conference on Software Engineering Theory and Practice, pp. 221-227, 2007.

[201]. Tang, L., J. Dong and T. Peng, A Generic Model of Enterprise Service-Oriented Architecture, 4th IEEE International Symposium on Service-Oriented System Engineering (SOSE), pp 1-7, December 2008.

[202]. Tang, L., Y. Zhao and J. Dong, Specifying Enterprise Web-Oriented Architecture, in High Assurance Services Computing, Springer, pages 241-260, 2009

[203]. Tang, L., J. Dong, T. Peng and W. T. Tsai, A Classification of Enterprise Service-Oriented Architecture, 5th IEEE International Symposium on Service-Oriented System Engineering (SOSE), pp. 74-81, 2010

[204]. Tang, L., J. Dong, T. Peng and W. T. Tsai, Modeling Enterprise Service-Oriented Architectural Styles, Service Oriented Computing and Applications (SOCA), Springer-Verlag, Vol. 4, p. 81-107, 2010

[205]. Tang, L., J. Dong, Y. Zhao and Liang-Jie Zhang, Enterprise Cloud Service Architecture, The 3rd IEEE International Conference on Cloud Computing, July 5 – 10, pp.27-34, 2010.

[206]. Tang, L., J. Dong and Y. Zhao, SLA-Aware Enterprise Service Computing, Chapter 2 in "Performance and Dependability in Service Computing: Concepts, Techniques and Research Directions, Edited by V. Cardellini, E. Casalicchio, K. C. Branco, J. Estrella and F.J. Monaco, IGI Global, p. 26-52, July 2011

[207]. Tang, L., F.B. Bastani, W. T. Tsai, J. Dong and L-J. Zhang, Modeling and Analyzing Cloud Service Architecture, Tech. Rep. UTDCS-26-11, Dept. of Computer Science, Univ. of Texas at Dallas, Sept. 2011

[208]. Taylor, H., A. Yochem, L. Phillips and F. Martinez, Event-Driven Architecture, Addison-Wesley, 2009

[209]. Taylor, R.N., N. Medvidovic and E.M. Dashofy, Software Architecture: Foundations, Theory, and Practices, Wiley, 2009

[210]. Ten-Hove, R. and P. Walker, "Java Business Integration (JBI) 1.0", Final Release, Sun Microsystems, Inc., 2005.

[211]. Tosic, V., K. Patel, and B. Pagurek, WSOL – Web Service Offerings Language. *Lecture Notes in Computer Science* , Vol. 2612, pp. 57-67, 2002.

[212]. Tosic, V., B. Pagurek, K. Patel, B. Esfandiari and W. Ma, Management applications of Web Service Offerings Language (WSOL), *Information Systems*, Vol. 30, No. 7, pp.564-586, 2005.

[213]. Tsai, W T., Service-Oriented System Engineering: A New Paradigm, Proceedings of the IEEE International Workshop on Service-Oriented System Engineering (SOSE), 2005, pp. 3-6

[214]. Tsai, W.T., C. Fan, Y. Chen, R. Paul, and J. Y. Chung, "Architecture Classification for SOA-based Applications", in Proc. of IEEE 9th International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC), April 2006, pp. 295-302.

[215]. Tsai, W.T., X. Bai and Y. Chen, Introduction to Service-Oriented System Engineering, TsingHua University Press, 2008.

[216]. Tsai, W. T., et al., "Semantic Interoperability and its Verification and Validation in C2 Systems", the 10th International Command and Control Research and Technology Symposium (ICCRTS), 2005, McLean, VA.

[217]. Tsai, W.T., et al, "Automatic Test Case Generation for GUI Navigation", Quality Week, 2000.

[218]. Tsai, W.T., W. Song, R. Paul, Z. Cao and H. Huang, Service-Oriented Dynamic Reconfiguration Framework for Dependable Distributed Computing, in Proceedings of the 28th Annual International Computer Software and Applications Conference, 2004

[219]. Tsai, W.T., Q. Huang, J. Xu, Y. Chen and R. Paul, Ontology-based Dynamic Process Collaboration in Service-Oriented Architecture, in Proceedings of the IEEE International Conference on Service-Oriented Computing and Applications, 2007

[220]. Tsai, W.T., X. Sun and J. Balasooriya, Service-Oriented Cloud Computing Architecture. The Seventh International Conference on Information Technology (pp.684-689).

[221]. Tsai, W.T., X. Sun and J. Elston, Real-Time Service-Oriented Cloud Computing. The 6th World Congress on Services (pp.473-478).

[222]. Varia, J. Cloud Architecture, 2008, http://developer.amazonwebservices.com/connect/entry.jspa?externalID=1632

[223]. Vinoski, S., REST Eye for SOA Guy, 11(1):82-84, 2007 IEEE Internet Computing

[224]. Vitvar, T., J. Kopecky, J.Viskova and D. Fensel, WSMO-Lite Annotations for Web Services, LNCS Vol. 5021, pp. 674-689, 2008

[225]. Vogels, W. Seamlessly Extending the Data Center - Introducing Amazon Virtual Private Cloud, Blogs, http://www.allthingsdistributed.com/2009/08/amazon_virtual_private_cloud.html

[226]. Vouk, M. A. Cloud Computing – Issues, Research and Implenetations, Proceedings of the 30th International Conference on Information Technology Interfaces, p. 31-40, June 23-26, 2008.

[227]. W3C, Web Services Eventing (WS-Eventing), http://www.w3.org/Submission/WS-Eventing/, 2006

[228]. W3C, "OWL-S: Semantic Markup for Web Services", http://www.w3.org/Submission/OWL-S/, 2004.

[229]. W3C, Web Services Description Language (WSDL) Version 2.0 Part 1: Core Language, W3C, 2007, http://www.w3.org/TR/wsdl20/

[230]. W3C, Web Services Choreography Description Language Version 1.0, W3C, 2005, http://www.w3.org/TR/ws-cdl-10/

[231]. W3C, Web Service Management: Service Life Cycle, W3C, 2004, http://www.w3.org/TR/wslc/

[232]. W3C, "Web Service Description Language (WSDL), Non-normative version with Z notation", August 2005.

[233]. Wang, G., C. Wang, A. Chen, H. Wang, C. Fung, S. Uczekaj, Y.-L. Chen, W. Guthmiller and J. Lee, Service Level Management using QoS Monitoring, Diagnostics, and Adaptation for Network Enterprise Systems. *Proceedings of the Ninth IEEE International EDOC Enterprise Computing Conference* (pp. 239-250, 2005.

[234]. Wang, H., G. Wang and C. Wang, A. Chen and R. Santiago, Service Level Management in Global Enterprise Services: from QoS Monitoring and Diagnostics to Adaptation, a Case

Study. *Proceedings of the Eleventh International IEEE EDOC Conference Workshop,* pp. 44-51, 2007.

[235]. Wikipedia, List of Web Service Specifications, http://en.wikipedia.org/wiki/List_of_Web_service_specifications

[236]. Wikipedia, Google App Engine, http://en.wikipedia.org/wiki/Google_App_Engine

[237]. Wikipedia, Measure Network Throughput, http://en.wikipedia.org/wiki/Measuring_network_throughput

[238]. Woods, D. and T. Mattern, Enterprise SOA, O'Reilly, 2006 (SAP)

[239]. Yan, S.S. and H. An, Adaptive resource allocation for service-based systems. Proceedings of the First Asia-Pacific Symposium on Internetware, 2009.

[240]. Yeom, G., W.-T. Tsai, X. Bai and D. Min, Design of a Contract-Based Web Services QoS Management System. *Proceedings of the 29th IEEE International Conference on Distributed Computing Systems Workshops*, pp. 306-311, 2009.

[241]. Zdun, U., C. Hentrich and W.M.P. van der Aalst, "A Survey of Patterns for Service-Oriented Architectures", International Journal of Internet Protocol Technology, vol. 1, no. 3, 2006, pp. 132-143.

[242]. Zeng, L., B. Benatallah, A. H.H. Ngu, M. Dumas, J. Kalagnanam and H. Chang, QoS-Aware Middleware for Web Services Composition. *IEEE Transactions on Software Engineering*, Vol. 30, No. 5, pp. 311-327, 2004.

[243]. Zhang, Liang-Jie and Qun Zhou, CCOA: Cloud Computing Open Architecture, IEEE International Conference on Web Services, p. 607-616, 2009.

[244]. Zhang, Liang-Jie, Jia Zhang and Hong Cai, Services Computing, Springer, Oct. 2007

[245]. Zhang, Liang-Jie, Carl K. Chang, E. Feig and R. Grossman, Keynote Panel, Business Cloud: Bringing the Power of SOA and Cloud Computing, pp. Xix, 2008 IEEE International Conference on Services Computing (SCC 2008), July 2008.

[246]. Zhang, Liang-Jie, Haifei Li and Herman Lam, Towards a Business Process Grid for Utility Computing, IT Professional, V. 6, N. 5, P. 63-64, 2004.

[247]. Zhang, Z., D. Dey and Y. Tan, Price and QoS competition in communication services. European Journal of Operational Research, Vol.186 i2, pp. 681-693, 2006.

[248]. Zhao, Y., J. Dong, and T. Peng, Ontology Classification for Semantic Web Based Software Engineering, IEEE Transactions on Services Computing. vol. 2, no. 4, pp. 303-317, Oct.-Dec. 2009.

[249]. Zhou, C., L.-T. Chia and B.-S. Lee, DAML-QoS Ontology for Web Services, *Proceedings of the IEEE International Conference on Web Services (ICWS'04)*, pp.472-479, 2004.

[250]. Zhou, J. and E. Niemela, Toward Semantic QoS Aware Web Services: Issues, Related Studies and Experience. *Proceedings of the 2006 IEEE/WIC/ACM International Conference on Web Intelligence*, pp. 553-557, 2006.

VITA

Longji Tang was born in Hunan Province, China. After graduating from Hunan University with a Bachelor of Engineering degree in Electrical Engineering in 1980, he worked as an associate research fellow at the Hunan Computing Center from 1980 to 1992. He began graduate studies at Penn State University in 1992 and graduated in 1995 with a Master of Engineering degree in Computer Science & Engineering and a Master of Art degree in Applied Mathematics. He has published more than 20 research papers from numeric analysis to computer applications in Journal of Computational Mathematics, Acta Mathematica Scienia and other publications. From 1995-2000, he worked as an information system and software engineering consultant at Caterpillar and IBM. Currently, he serves as a senior technical advisor at FedEx's information technology division and has undertaken his PhD studies in Software Engineering as a part-time student at the University of Texas at Dallas since June, 2002. He obtained PhD degree from Computer Science of UTD in 2011. At FedEx he has served as a tech lead and/or architect on several critical eCommerce projects and is currently a lead project manager for FedEx.com's data center modernization project. His research interests include software architecture and design, service-oriented architecture, service-oriented cloud computing and application, and system modeling and formalism.