High Assurance Services Computing

Jing Dong · Raymond Paul · Liang-Jie Zhang
Editors

High Assurance Services Computing

Springer

Editors
Jing Dong
Department of Computer Science
University of Texas, Dallas
2601 N. Floyd Road
P.O. Box 830688
Richardson TX 75083
USA
jdong@utdallas.edu

Liang-Jie Zhang
IBM Research
19 Skyline Dr.
Hawthorne NY 10532
USA
zhanglj@us.ibm.com

Raymond Paul
Department of Defense
4502 7th St. NE.,
Washington DC 20017
USA
raymond.paul@osd.mil

ISBN 978-0-387-87657-3 e-ISBN 978-0-387-87658-0
DOI 10.1007/978-0-387-87658-0
Springer Dordrecht Heidelberg London New York

Library of Congress Control Number: 2009928694

Table of Contents

Preface

Services computing is an emerging discipline cross-cutting the science, engineering and technology. It bridges the gap between Business Services and IT Services. The scope of services computing covers the whole lifecycle of services innovation research and practice that includes services modeling, creation, deployment, discovery, composition, analysis, and management. The goal of services computing is to facilitate the application of loosely-coupled services and computing technology for building systems more efficiently and effectively. The core technology suite includes Service-Oriented Architecture (SOA) and Web services. SOA is a common platform for implementing large scale distributed applications by composing services, which are platform independent components running on different hosts of a network. It offers native capabilities, such as publication, discovery, selection and binding for creating new applications by combining services as basic building blocks. A repository of existing services independent of the underlying infrastructures can be discovered and composed in an application. The requester and the provider exchange messages via the network through standard protocols.

SOA is now being deployed in mission-critical applications in domains that include space, health-care, electronic commerce, telecommunication, and military. Many critical systems require multiple high assurance, including reliability, safety, dependability, security, and availability. Failures of such systems may cause the loss of human lives and finance. For example, the reliability of aircraft/spacecraft navigation and guidance control systems can affect human lives; the correctness and timeliness of military command and control systems can be crucial to the success of defense missions; the failure of a medical process-control system can cause death or injury to the patient; the failure of a banking system can cause property losses for many clients; the failure of a security management system in a network server can cause chaos and result in financial or intellectual property losses; the failure of railroad control systems can cause delays and subsequent financial losses or can even lead to catastrophic life threatening failures. In modern human society, our reliance on computer systems can be observed in our daily lives. From the current trend, our reliance on high assurance systems will grow at an increasing pace. Thus, there is a pressing need for developing computer systems whose quality can be guaranteed to a high degree; otherwise, we will risk the well-being of societies at the hands of computer hardware and software failures or misuses by human intruders. Existing methods dealing with such constraints may be not readily applied in service-oriented environment. Different from traditional computer-based systems, services are typically third-part entities. There is no standard way to define high assurance properties in service specifications. Service interfaces normally focus on the descriptions of functional aspects, such as input, output, pre/post conditions (IOPE). The high assurance properties of a service are generally unclear or defined in an ad hoc manner in the service interfaces. This poses new challenges on service discoveries with high assurance requirements.

A successful service needs to provide the required functionality and the necessary Quality of Service (QoS). The QoS parameters are typically specified in service level agreements (SLAs) that the service provider needs to guarantee and their violation will be penalized appropriately. The QoS constraints that a service provider guarantees may include run-time properties, such as timeliness, transaction rate, and availability, as well as design-time properties, such as language of service and compliance. Such high assurance guarantees are difficult to ensure when services are spatially distributed over a network subject to active attacks, network congestion, and link delays, which may pose a formidable challenge in delivering services that meet the SLAs.

There are a number of important issues in high assurance services computing:

- How to describe, assess, and ensure Quality of Service in service-oriented systems?
- How to manage and evaluate dependability of service compositions from individual services?
- How to analyze and assess the trustworthiness of service requestors and service providers?
- How to facilitate service creations and executions?
- How to verify service behavior and service level agreement?
- How to engineer service-oriented systems?
- How to test service applications?

This book is a collection of fourteen chapters solving some of these problems.

About This Volume

Chapter 1 defines separate levels of Quality of Service (QoS) assurance within a service-oriented architecture. Each of these levels includes replication options that can bring substantial benefits toward high assurance of run-time related non-functional properties (NFP) in complex environments. Experimental results based on architectural translucency in health care applications showed an increase of 50% on the NFP levels with more stable QoS levels. The NFP representation has been formalized for automating runtime assurance and matching between required and provided QoS levels. System reconfiguration techniques for the different levels within an SOA will dynamically adapt the architecture so that it provides QoS assurance at different loads.

Chapter 2 considers the challenges of assessing highly critical net-centric systems. A trustworthiness ontology is developed to capture the trustworthiness aspects and their correlations as well as to model various classes of system entities and their integrations. The ontology provides information to guide the trustworthiness analysis and data collection. Based on the ontology, a trustworthiness assessment framework is developed. In the framework, systematic steps are formulated to achieve trustworthiness assessments. Techniques and tools to perform the assessments in each step are incorporated in the ontology to allow the actual analysis and derivation of assessment results. A holistic assessment technique is developed to provide a single overall measure of the trustworthiness of a system or a subsystem.

Chapter 3 presents a monitoring architecture for managing trust rules in service interactions. The trust rules identify the contexts of trust concerns and snapshot system events encapsulating a service outcome that is crucial to the target system. The proposed architecture, called Trust Architecture for Monitoring, may reside in each service provider, which allows the analysis of the trustworthiness of users based on trust rules and calculation schemes. A service requestor is penalized for the violation of trust rules and rewarded otherwise, which thus facilitates the quantification of its trustworthiness. Incorporating the recommendations from similar service providers may help collaborative decision making. The performance overhead of the architecture has been evaluated based on the monitoring of a prototype trust-aware file-sharing grid.

Chapter 4 addresses the key policy challenges of human interoperability enterprise (HIE) and highlights major steps that can lead to the development of a holistic interoperability policy framework for engineering high-assurance systems. The human performance criteria for high-assurance and trustworthy systems are elaborated. The HIE systems are designed by integrating core technology components and methodologies drawn from the area of human cognitive engineering. The key challenges and elicit solutions of HIE systems are closely related to the technological areas including Human-Centered Computing, Information, Knowledge and Intelligence Management, service-oriented architecture, and behavioral sciences.

Chapter 5 describes the architecture of the Service Execution Environment that hides the complexity of the communication environment and the Service Creation Environment to help service developer in evaluating the quality of an orchestration of telecom-IT services. Both static and dynamic non-functional properties are aggregated by the Aggregator service that calculates the overall aggregated non-functional properties of a service composition designed by the developer, relying also on the Monitor manager which provides live values of dynamic non-functional properties such as response time.

Chapter 6 introduces a performance measurement framework for cyberphysical systems. The framework includes a cyberspatial reference model for establishing the identity and location of servers and clients in distributed high-assurance service systems. It also defines a set of service performance indices to measure the reliability, availability, safety, security and timeliness properties. An application neutral, yet operational definition of value useful in high assurance service systems is developed for defining their respective value propositions.

Chapter 7 applies graph grammars for verifying the behavior of service-oriented systems. The behavior verification problem is cast to a visual language parsing problem. A behavior graph is parsed with user-specified rule-based constraints/properties expressed by a graph grammar. A parsing result indicates whether the observed behavior satisfies its requirements or not. A parsing error represents a potential problem in the service behavior. The approach allows developers to check the acceptable sequence of message exchanges between services confirming to some requirements/specifications.

Chapter 8 provides a distributed service-oriented asynchronous framework in an event-driven formal synchronous programming environment. This model-driven framework is based on a synchronous programming language SOL (Secure Operations Language) that has capabilities of handling service invocations asynchronously and provides strong typing to ensure enforcement of information flow and security policies. The clients' requirements and the service level agreements can be ensured in the service-oriented systems that have been formally verified. An infrastructure for deploying and protecting time- and mission-critical applications on a distributed computing platform is developed especially in a hostile computing environment, such as the Internet, where critical information is conveyed to principals in a manner that is secure, safe, timely, and reliable.

Chapter 9 offers a coordination model for building dynamically adaptive service oriented systems. Each service is situated in and coordinated by an active architectural context, which mediates the interactions among the services. The architecture of service oriented applications is self-adaptive for bridging the gaps between environment, system and application goals with an ontology-based approach. An access control model is proposed for secure service coordination logic as well as keeping service autonomy discretionarily with a decentralized authorization mechanism. Three classes of trust relationships are also identified for a trust management framework to help the understanding and assurance of the trustworthiness of service oriented applications.

Chapter 10 develops a generalized and comprehensive framework to evaluate and maximize diversity for general service-oriented systems. The dependability attributes of individual service components under diverse operational conditions are evaluated. The internal assessments of services are linked to their external dependability attributes. The preferences of a specific set of stakeholders can also be used to assess the relative importance and trade-off among dependability attributes. The evaluation framework also includes an overall methodology that maximizes system diversity using a mathematical optimization technique for ensuring system dependability via diversity maximization that combines collective strengths of individual services while avoid, complement, or tolerate individual flaws or weaknesses.

Chapter 11 transforms the BPEL processes into Unified Modeling Language (UML) sequence diagrams for consistency analysis. Since sequence diagrams are intuitive and show temporal-based execution naturally, they help to ease the learning curve of BPEL's nomenclature and reduce errors. Two examples have demonstrated the discovery of certain errors in the sequence diagrams with tool support.

Chapter 12 specifies both structurally and behaviorally the Enterprise Web-Oriented Architecture (EWOA) and analyzes its software quality attributes. The specification of the EWOA is based on a generic model of the Enterprise Service-Oriented Architecture. The EWOA style consists of a set of design principals based on REST and Web 2.0, a set of architectural elements of infrastructure, management, process, and a set of software quality attributes. Based on the analysis of the security and manageability issues of EWOA, the pure RESTful system architecture with RESTful QoS governance and a hybrid approach with both REST and SOAP for enterprise are proposed.

Chapter 13 outlines a service oriented architecture for the Peer-Assisted ContenT Service (PACTS) that is a video on demand streaming system. The PACTS organizes elements of traditional video streaming and peer to peer computing into loosely-coupled composable middleware services and distributing them among participating entities for high-quality low-cost video streaming at a large scale and in real time. The implementation of PACTS has demonstrates effectively offload server's bandwidth demand without sacrificing the service quality and in dynamic settings with system churns. It shows significantly reduces bandwidth utilization at the server by leveraging peer assistance. The service level agreement specification is modeled to differentiate QoS to end users based on their bandwidth contributions to the system to derive the minimum and maximum QoS level given a bandwidth budget at the server side.

Chapter 14 proposes a Model-based Adaptive Test (MAT) for multi-versioned software based on the Coverage Relationship Model (CRM) for case selection and ranking technique to eliminate redundant test cases and rank the test cases according to their potency and coverage. It can be applied in various domains, such as web service group testing, n-version applications, regression testing, and specification-based application testing. Two adaptive test cases ranking algorithms are provided by using the coverage probability. Experiments are conducted using the proposed techniques. The experiment results indicate that the CRM-based test case selection algorithm can eliminate redundant test cases while maintaining the quality and effectiveness of testing.

This book is intended particularly for practitioners, researchers, and scientists in services computing, high assurance system engineering, dependable and secure systems, and software engineering. The book can also be used either as a textbook for advanced undergraduate or graduate students in a software engineering or a services computing course, or as a reference book for advanced training courses in the field.

Acknowledgements

We would like to take this opportunity to express our sincere appreciation to all the authors for their contributions and cooperation, and to all the reviewers for their support and professionalism. We are grateful to Springer Publishing Editor Susan Lagerstrom-Fife and her assistant Sharon Palleschi for their assistance in publishing this volume.

Jing Dong
Raymond A. Paul
Liang-Jie Zhang

Chapter 1

Translucent Replication for Service Level Assurance

Vladimir Stantchev*[1] and Miroslaw Malek**

* International Computer Science Institute, Berkeley, California (vstantch@icsi.berkeley.edu)

** Humboldt-University at Berlin, Germany

Abstract: Web services are emerging as the technology of choice for providing functionality in distributed computing environments. They facilitate the integration of different systems to seamless IT supporting infrastructure for business processes. Designing a service-oriented architecture (SOA) for this task provides a set of technical services and composition techniques that offer business services from them. There are two basic aspects of a successful service offering: to provide the needed functionality and to provide the needed Quality of Service (QoS). Mission-critical applications in health care require high and stable QoS levels. The complexity of different web service platforms and integration aspects make the high assurance of such run-time related nonfunctional properties (NFPs) a non-trivial task. Experimental approaches such as architectural translucency can provide better understanding of optimized reconfigurations and assure high and stable QoS levels in mission-critical clinical environments.

1. Introduction

Web services are emerging as a dominating technology for providing and combining functionality in distributed systems. A service-oriented architecture (SOA) offers native capabilities, such as publication, discovery, selection and binding [1]. Since services are basic building blocks for the creation of new applications, the area of composite services is introduced on top of native capabilities. It governs the way applications are developed from basic services. Here, richer interface de-

[1] Vladimir Stantchev is also a senior lecturer at the Fachhochschule fuer Oekonomie und Management in Berlin, Germany

finitions than the Web Service Description Language (WSDL) are needed and they can be provided in the form of contracts [2, 3].

There are two basic aspects of a successful service offering: to provide the needed functionality and to provide the needed Quality of Service (QoS). QoS parameters are part of the nonfunctional properties (NFPs) of a service, typically specified in service level agreements (SLAs). We distinguish between runtime related and design-time related NFPs. Run-time related NFPs are performance oriented. Examples are response time, transaction rate, availability. Design-time related NFPs such as language of service and compliance are typically set during design time and do not change during runtime. Run-time related NFPs can change during runtime when service usage patterns differ (times of extensive usage by many users are followed by times of rare usage), or when failures occur. Such failures can occur within the service, as well as in the network components that lie between user and service. NFPs and QoS are regarded (together with semantics) as topics that encompass all three levels of services within an SOA (basic services, composite services, managed services) [1].

Formalization and specification of NFPs and their SLAs is currently a very active research field. The enforcement of these levels for runtime-related NFPs cannot be done automatically a priori, due to the changes in service usage and network availability. An approach to dynamically adapt service performance to these changes can ensure continuous meeting of service levels. Providing such dynamically reconfigurable runtime architectures is regarded as one of the main research challenges in the area of service foundations [1]. Such approach should employ service reconfiguration at runtime, as changes in source code of a service are not a feasible option. One approach to identify possible reconfigurations in an SOA and evaluate their implication is called architectural translucency [4]. It describes the notion that different levels in an SOA can have different implications to service levels of NFPs and that understanding these implications is key to provide service level assurance. A central aspect of this approach is to evaluate different replication configurations at the operating system (OS) and serviceware (SW) level and how they affect web service performance.
Health care applications often require high and stable QoS levels. This is particularly true for clinical environments where mission-critical IT systems support lifesaving activities.

In order to apply architectural translucency to address high assurance of NFPs in such clinical environments, several questions arise. First, what is the relation between replication and assured service levels, especially concerning runtime related NFPs (Section 2) and how can we formally represent performance aspects (Section 3). Second, what methods are well suited to research this relation and to recommend optimized replication configurations (Section 4). Finally, what are the possibilities to integrate automated assurance of service levels in a clinical environment based on these recommendations (Section 5).

2. Service Level Assurance of Performance

This section describes the effect of web service replication on performance, presents architectural translucency as approach to decide optimized reconfigurations and the importance of the OS and SW levels as places for possible replications.

2.1 Replication and Performance

Performance, more specifically transaction rate, is defined as the system output $\omega(\delta)$ that represents the number of successfully served requests from a total of input $\iota(\delta)$ requests during a period of time δ. This is a generalized view of the equation model presented in [5], where it is referred to as throughput *Xa*.

$$\omega(\delta) = f(\iota(\delta))$$

The performance of a serial composed service chain is determined by the service with the lowest performance. If that service is Service N then its performance can be defined as follows:

$$\omega^{ServiceN} = f^{ServiceN}(\iota_1)$$

The performance of a serial composed service chain that includes Service N would be:

$$\omega^{ServiceChain} \leq f^{ServiceN}(\iota_1)$$

The performance of replicated composed service chain that includes Service N would be:

$$\omega^{ServiceChain} \leq 2 * f^{ServiceN}(\iota_1)$$

This definition corresponds to transaction rate as NFP.

Another typical run-time related NFP is response time. The average response time can be derived from the transaction rate as follows:

$$RT_{avg} = \frac{1}{\omega}$$

Therefore, replication has advantageous effects on service chain performance when no replica synchronization is required. This applies to transaction rate and response time as NFPs in a SOA.

The traditional view of availability is as a binary metric that describes status. Status can be "up" or "down" at a single point of time. A well-known extension is to compute the percentage of time, on average, that a system is available during a certain period. This results in statements where a system is described as having 99.99% availability, for example.

There are several extended definitions of availability that address the inherent limitations of this definition – availability should be considered as a spectrum, rather as a binary metric. It should also reflect QoS aspects. One possibility is to measure availability by examining variations in system QoS metrics over time [6]. Therefore, assurance of stable QoS metrics leads to better availability.

2.2 Architectural Translucency

The complexity involved in providing a single web service is often underestimated. A look at hardware platforms, even commodity hardware, reveals complex microprocessors and processing architecture. Standard OSs are far away from microkernel designs and contain a large number of OS extensions. These are called *modules* in a Linux system and *drivers* in a Windows system. Beside typical device drivers, extensions include network protocol implementations, file systems and virus detectors. Typical component environments such as .NET and J2EE often serve as the middleware for providing web services [7], here referred to as serviceware. A look at the application programming interfaces of these environments reveals their complexity.

One general problem in such complex environments is where to introduce a certain measure (e.g., replication), so that the system can assure optimized performance at certain loads.

Much work has been done in the area of QoS-aware web service discovery [8], QoS-aware platforms and middleware [9,10,11,12], and context-aware services [13]. However, all of these approaches do not address assurance of service levels by a single service, but rather deal with the composition of services where aggregated NFP levels would satisfy a specific requirement.

The existing standards for specification of QoS characteristics in a service-oriented environment can be grouped according to their main focus: software design/process description (e.g. UML Profile for QoS and QML - QoS Modeling

Language) [14], service/component description (e.g. WS-Policy) and SLA-centric approaches (e.g. WSLA - Web Service Level Agreements [15], WSOL - Web Service Offerings Language [16], SLAng - Service Level Agreement definition language [17] and WS-Agreement [18]).

Extensive research concerning NFPs exists in the field of CORBA (Common Object Request Broker Architecture), particularly in the areas of real-time support [19,20], replication as approach for dependability [21,22,23,24], adaptivity and reflection [25,26], as well as mobility [27,28]. Similar approaches involving replication have been proposed for J2EE-based web services [29,30,31].

To the best of the authors' knowledge, there are no other published works that address the question where and how an optimized reconfiguration can be introduced in the complex of hardware, OS and component environment in order to optimize the NFPs of web services. Of particular interest is to evaluate whether reconfigurations at one level are generally more advantageous than others. This is the main objective of architectural translucency as an approach for service level improvement and assurance. The approach is an extension of architectural approaches that aim to improve NFPs in one location, e.g., reliability at the OS level [32], scalability by clustering of web servers [33] or email servers, as well as introducing software RAID approaches [6]. Architectural translucency defines levels that encompass these approaches and compares replication configurations at the different levels. These levels are: hardware, operating system and serviceware.

Failures at the network level lead to network partitions. There is currently no convincing way to mathematically model network partitions [34]. Furthermore, it is NP-hard to derive a partition model from link and node failure models [35]. Currently, architectural translucency does not address questions of network availability and performance. Nevertheless, there are several promising approaches that can be combined with architectural translucency in order to incorporate network availability in overall availability of distributed systems. One possible way is to incorporate network failures in availability metrics that define $Avail_{client} = Avail_{network} \times Avail_{service}$ [34]. Better assignment of object replicas to nodes can further improve availability in such settings [36].

2.3 Experimental Computer Science

Architectural translucency can be classified in the field of experimental computer science [37]. There are three key ideas in experimental computer science – a hypothesis to be tested, an apparatus to be measured, and systematic analysis of the data to see whether it supports the hypothesis [37]. The hypothesis is that replications at different levels and in different ways have different effect on web service run-time related NFPs (specifically performance). The apparatus consists of typical platforms for web services (Windows Server with .NET and Internet Information Server (IIS), UNIX with WebSphere) and web service benchmarks. Tools like

Microsoft Application Center Test (ACT) and HP LoadRunner allow for automated testing of web services during long periods and with different loads. They also facilitate the gathering of large amounts of test data. Statistical tools such as SPSS [38] and R [39] are well suited to further analyze this data.

3. Performance Models

Some approaches to model performance-related aspects of a system are described in [40,41,42,43]. One promising approach to analytically model multi-tier Internet applications was recently published in [44]. The model is based on a network of queues where the queues represent different application tiers. This research effort is similar to efforts of Kounev and Buchmann [45] and, more recently, Bennani and Menasce [46]. The second work is based on previous research published in [47,5,48,49,50]. Kounev and Buchmann also use a network of queues to predict performance of a specific two tier application and solve the model numerically using existing analysis software. Bennani and Menasce model a multi-tier Internet service that serves multiple types of transactions. The model is again based on a network of queues with customers belonging to multiple classes.

There are some other recent efforts to model multitier applications. These are often extensions of single-tier models. One approach [51] considers server provisioning only for the Java application tier and uses an M/G/1/PS model for each server in this particular tier. Another approach [52] models the same tier as a G/G/N queue. Other works have modeled an entire multi-tier application using a single queue (e.g., a M/GI/1/PS queue in [53]).

Various works describe complex queuing models. Such models can capture simultaneous resource demands and parallel subpaths within a tier of a multitier application. One example is Layered Queuing Networks (LQNs). They are an adaptation of Extended Queuing Networks which account for the fact that software servers run atop of other layers of servers, thus giving complex combinations of simultaneous resource requests [54,55,56,57,58]. The focus of these works lies primarily on Enterprise Java Beans-based application tiers.

Extensive work exists in the area of modeling of single-tier Internet applications, most commonly HTTP servers. One of the early works [59] introduced a network of four queues to model static HTTP servers. Two of the queues model the web server, and two – the Internet communication network. Another approach [60] also presented a queuing model and related average response time to available resources. A GPS-based (Generalized Processor Sharing) queuing model of a single resource (e.g., CPU) at a web server was proposed in [61,62]. A G/G/1 queuing model was suggested in [63], a M/M/1 queuing model to compute web request response times – in [64]. One web server model with performance control as objective was introduced in [65]. Menasce [66] presented in 2003 a combination of a Markov chain and a queuing network model, an idea originally presented in [67]. Despite these tremendous developments such models still cannot fully reflect the complexity of the three layers of web service platforms concerning NFPs. Therefore, experimental methods can help to further enhance our knowledge of optimized configurations in such complex settings.

4. Translucent Replication

In service-oriented computing (SOC) a node receives a stream of requests, processes them and sends back a stream of results. From an operating system point of view there exist two general strategies for request processing – threaded request processing and event-driven request processing. There are two questions that architectural translucency can address in this context:

1. Are there ways to introduce (or alter default) replication settings at the OS and SW level?
2. Can a system assure optimized performance (or other QoS) by reconfiguring such OS or SW settings?

4.1 Threaded Request Processing

When implementing pure threaded request processing an OS creates a thread for each client request. The whole request and maybe its subsequent ones from the same client are processed in the context of this newly created thread. This approach offers good parallelization of request processing and good utilization of available resources. Main disadvantage is the large overhead of the thread lifecycle (creation, management, deletion). Modern OSs address this by employing thread pools – a specific number of request processing threads are created in advance and left idle. The OS can then dynamically assign them to incoming requests. Implementations differ in handling new requests if all worker threads are busy.

4.2 Event-driven Request Processing

With pure event-driven request processing the OS processes all requests within a single worker thread. Arriving requests are stored in a queue which is used as input for the worker thread. The worker fetches and processes the requests one at a time. The per request overhead here is minimal (managing the request queue) and a queuing policy with request priorities can be introduced. The approach is nevertheless contradictory with a key goal of architectural translucency – to ensure high resource utilization. The low degree of parallelism can result in longer idle times.

4.3 Thread and Process Replication

At operating system level there are two general ways for functional replication – replication of threads and replication of processes. While in the former a node creates a new thread to process a client request, in the latter it creates a new process to handle an incoming request. Generally, it can be assumed that a per-thread replication should be more favorable to performance as per-process replication, as the overhead of process management is larger from an OS point of view. The research of this hypothesis requires knowledge about the mechanisms to control this type of replication at the OS level of typical platforms for web services.

4.4 Levels

The hypothesis is that there are different levels (HW, OS, SW) where a service provider can introduce replication and different ways to replicate at each level. Furthermore, there are differences in web service performance when the provider applies them. The objective is to define these different ways of replication and find reconfiguration techniques for them. High assurance during runtime often makes hardware changes unfeasible or costly. Therefore, in this chapter we focus on architectural translucency aspects at the OS and SW level.

4.4.1 Operating System Level

When working with typical OS/Middleware configurations for SOC (Windows Server 2003 with IIS 6 and the .NET Framework, or UNIX and IBM WebSphere) a service provider has to consider several specifics.

Pure threaded or event-driven implementations are rare. There are different design patterns for hybrid implementations (staged request processing, thread pools, reactor pattern).

Windows 2003 with IIS 6 uses the thread pools pattern – when deploying a web service in IIS 6 it creates one process for the service that also contains a number of precreated threads. While IIS 5 allowed only changing the number of these threads per web service process (number specified in the configuration file machine.config), IIS 6 allows also specifying the number of process replicas to serve multiple requests. Furthermore, IIS 6 ignores threading parameters known from IIS 5 such as maxWorkerThreads and minFreeThreads, as threading optimization is automated. A test of replication alternatives at the OS level involves specifying a higher number of process replicas for a web service (see Figure 1). WebSphere also uses the tread pools pattern; configuration settings are accessible via the *application server* menu. The menu item *Thread-Pools* contains an overview of the existing thread pools within the application server. By selecting *Web Container* the specific parameters of the thread pool can be configured. The settings for processes and the Java Virtual Machine (JVM) are accessible in the group *Java and Process Management*, menu *Process Definition*.

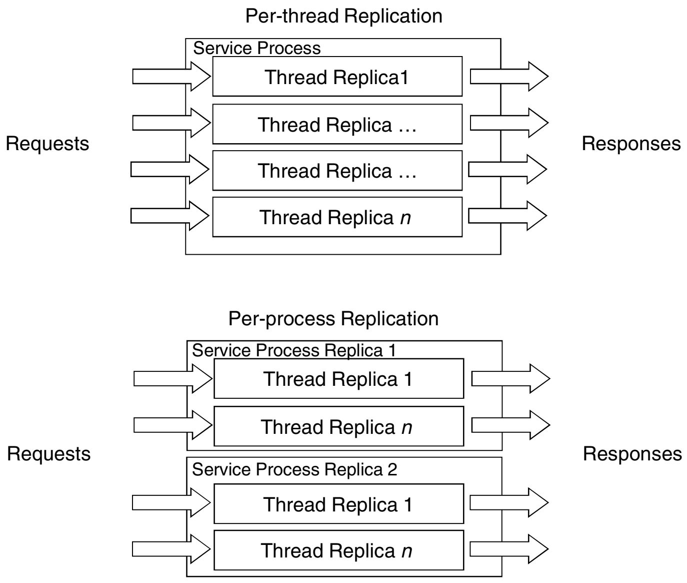

Fig. 1. Replication at OS Level: Per-process Replication vs. Per-thread Replication

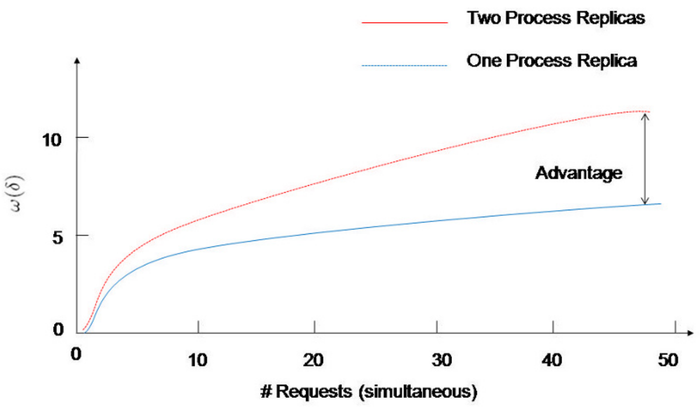

Fig. 2. Replication at OS Level: Advantage of Per-process Replication vs. Per-thread Replication

Tests have demonstrated that having two process replicas per web service instead of one can lead to throughput increases of up to 50% under higher loads (more than 40 simultaneous client requests, see Figure 2). This applies to services that are already optimized using asynchronous requests and minimizing need for exclusive hardware access (e.g., hard disk). Performance is also far more stable with confidence intervals of 99%.

4.4.2 Serviceware Level

Nodes in SOC typically use an application server to manage and host services. Such application server corresponds to the serviceware level of the presented approach. It simplifies generic service management tasks, e.g., service configuration, deployment, connection (with other services) or monitoring. These tasks are often done using service containers.

Services within a service container can be composed using two general structures: direct composition and composition via a service bus. Direct composition of services within a service container resembles the component programming paradigm: services are explicitly connected with other required services at deployment time. As precondition the required services must be available. The service bus concept connects all deployed services to a central bus which is responsible for request routing to the services. This allows more complex interactions such as the publish/subscribe approach known from enterprise application integration.

When looking at replication at the serviceware level there are two basic alternatives – replication of service containers (see Figure 3) or replication of services within service containers (see Figure 4).

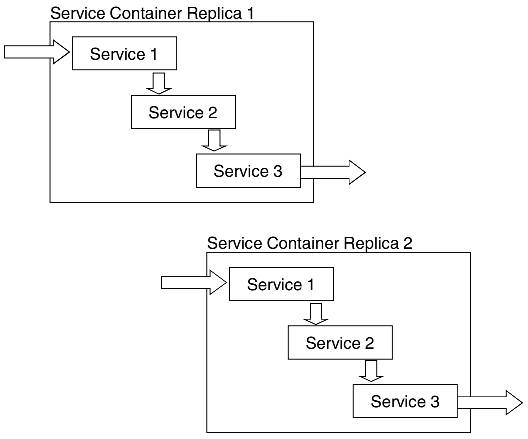

Fig. 3. Replication of a Service Container

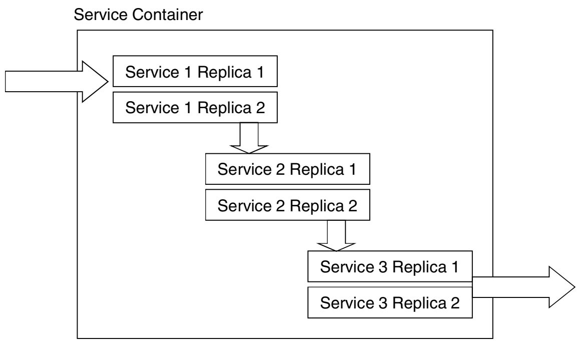

Fig. 4. Replication of Services within a Service Container

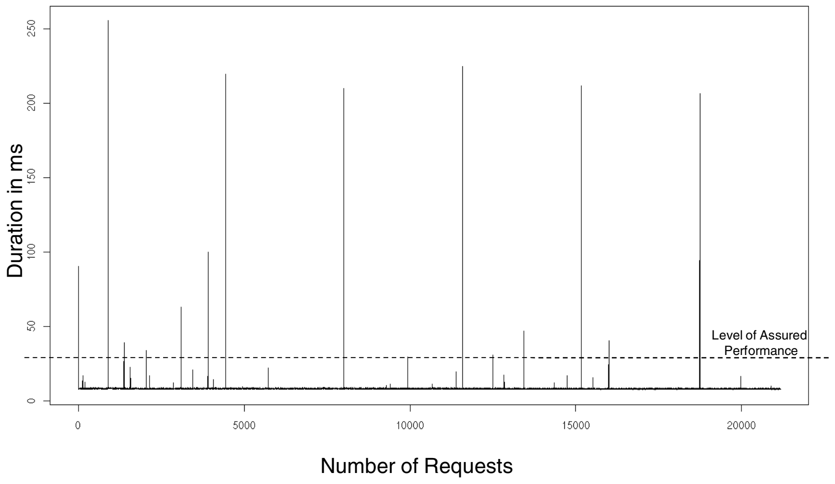

Fig. 5. High Assurance through Serviceware Replication

From an object-oriented point of view both these alternatives can be implemented by instantiating new service or container objects. An objective of architectural translucency is to allow for such reconfigurations without reprogramming.

When dealing with replication at the serviceware level using WebSphere, the question is how to distribute instances in different Web Containers (Web Containers serve as service containers in WebSphere). Possible ways are to use another main context or to change the main context manually within the EARs. Manual change is done by editing the file application.xml in the META-INF directory. The service provider has to edit the pairs of names so that there is no match within a pair, especially concerning the elements display-name, web-uri and context-root. Names of web archive (WAR) files also have to be adapted accordingly before packing the EAR with jar.

Our results here show distinct performance advantages for replication within a service container as compared to replication of service containers. Furthermore, when we focus on aspects of high assurance we observe substantially higher confidence intervals in performance stability, as shown exemplarily in Figure 5. The required response time (30ms) is assured for all but 16 requests from 22 000 requests overall, resulting in an assurance rate higher than 99.9 percent. Our framework deals also with these delayed requests by a resubmission after a certain timeframe expires.

5. High Assurance in the Operating Room

The application scenario focuses on the surgical sector. It is not only one of the largest cost factors in health care but also a place where failures to provide timely needed information can be perilous, endangering life and health of a patient.

Pre- and postoperative processes are key factor for the effective and safe utilization of the surgical sector.

5.1 Perioperative and Postoperative Processes

The perioperative processes start with a notification from an operating room nurse or an anesthesia nurse, that the staff should transport the next patient to the operating room. Then a transport service or a nurse moves the patient from the ward to the operating room area. In the main registration area clinicians transfer the patient from the ward bed to an operating room table. Afterward the patient resides in the induction area, where he is anesthetized. Then clinicians move the patient to the operating room, where the preparation for the operation starts, for example operation specific bedding, sterile coverage etc. The surgery starts with the cut and finishes with the suture. After the surgery clinicians transport the patient to the post anesthesia recovery unit, where he is moved again to the ward bed and recovers from anesthesia. After the recovery the staff transports the patient back to the ward.

There is an extensive usage and movement of things (devices, instruments, beds) related with these processes. Furthermore, such devices and instruments need a preparation (e.g., disinfection) prior to usage. Proximity of clinicians to such things typically indicates intended or current usage. Therefore, position information is a key input for the planning and steering process.

Furthermore, there are high requirements regarding performance and other NFPs that the IT infrastructure needs to satisfy.

5.2 Technology Environment and Architectural Approach

There exist a variety of position sensing systems than are suited for deployment in such environments [68]. An integration of such system, together with a hospital information system (HIS) and enterprise resource planning system (ERP) can provide the needed functionality to optimize surgical processes.

Figure 6 shows our integration approach within an SOA. Here the WLAN positioning system, the HIS and the ERP system are integrated in the SOA with wrappers that provide web service interfaces to the enterprise service bus (ESB). Clinicians are using Tablet PCs as mobile devices; Devices and patients are equipped with WLAN tags.

The usage of an SOA in such mission-critical environments depends heavily on the high assurance of run-time related NFPs. For example, data about position of monitored objects (more than 10000) has to be available within 5 seconds. The AT engine is responsible for service QoS assurance by monitoring and management. In a first step, it measures performance of services in their standard configurations at the OS and serviceware levels. We then import the QoS requirements and evaluate them. We presented a structure for their formalization in [69]. Using these formalized requirements, the AT engine configures the proper settings at each service platform. During runtime, when the engine notices that for example a service is experiencing higher loads, it dynamically reconfigures the replication settings of the service platform to further provide the expected QoS.

Representation and further information processing are depicted in the upper part of the figure. The system provides portal-based access to process-related information. Examples are electronic patient records (EPRs) or case definitions that are extracted from the HIS and visualized on the Tablet PC. Which patient record or case definition is visualized depends on the current location of the Tablet PC and otherWLAN-enabled objects that surround it (e.g., patient tags).
Furthermore, the system offers more complex planning, steering and evaluation functions. These are provided by composite services.

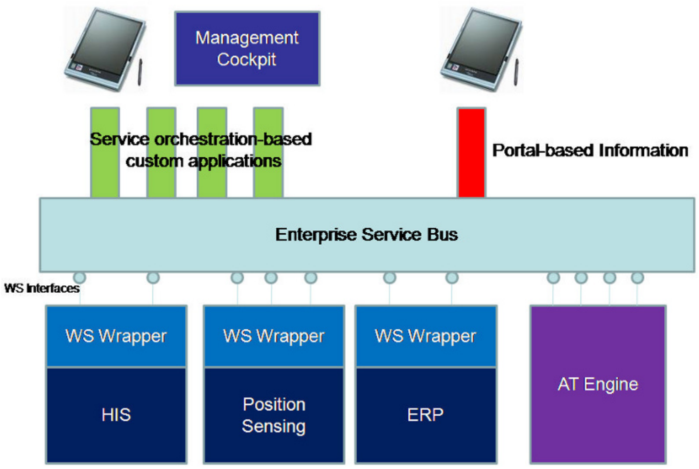

Fig. 6. Architectural view of the solution. HIS - hospital information system, ERP - enterprise resource planning system, AT - architectural translucency

6. Summary

Mission-critical environments in clinics require high assurance of performance and other run-time related NFPs. Typical platforms for providing web services are complex and hardly predictable. Seamless IT support of processes often requires integration of different *off-the-shelf* systems such as HIS and ERP. Location awareness can optimize usage planning, monitoring and steering of resources in clinical environments. Position sensing systems based on radio technology (e.g., RFID, WLAN) provide such information and are key components of clinical IT support. The design of an SOA is a promising approach to integrate these different systems. Such integration typically requires the development of web service wrappers around the interfaces of the systems and leads to further increases in complexity. This makes QoS assurance even more compelling. A definition and separation of levels within an SOA, as well as a look at replication options at these levels can bring substantial benefits toward high assurance of run-time related NFPs in such complex environments. Experimental approaches such as architectural translucency are well suited for this task and can increase assured NFP levels by 50%. They can also provide more stable QoS levels.

Automated run time assurance further requires formalization of NFP representation and matching between required and provided QoS levels. Furthermore, automated assurance systems need to provide integrated system reconfiguration techniques for the different levels within an SOA. Such run time reconfiguration will dynamically adapt the architecture so that it provides QoS assurance at different loads.

References

[1] Michael P. Papazoglou, Paolo Traverso, Schahram Dustdar, and Frank Leymann. Service-oriented computing: State of the art and research challenges. Computer, 40(11):38–45, Nov. 2007.
[2] Nikola Milanovic and Miroslaw Malek. Current solutions for web service composition. IEEE Internet Computing, 8(6):51–59, 2004.
[3] Francisco Curbera. Component contracts in service-oriented architectures. Computer, 40(11):74–80, Nov. 2007.
[4] Vladimir Stantchev and Miroslaw Malek. Architectural Translucency in Service-oriented Architectures. IEE Proceedings - Software, 153(1):31–37, February 2006.
[5] Daniel A. Menascé. QoS issues in Web services. Internet Computing, IEEE, 6(6):72–75, 2002.
[6] A. Brown and D.A. Patterson. Towards Availability Benchmarks: A Case Study of Software RAID Systems. Proceedings of the 2000 USENIX Annual Technical Conference, 2000.
[7] Gerry Miller. The web services debate: .net vs. j2ee. Commun. ACM, 46(6):64–67, 2003.
[8] Y. Makripoulias, C. Makris, Y. Panagis, E. Sakkopoulos, P. Adamopoulou, M. Pontikaki, and A. Tsakalidis. Towards Ubiquitous Computing with Quality of Web Service Support. Upgrade, The European Journal for the Informatics Professional, VI(5):29–34, 2005.
[9] S.S. Yau, Yu Wang, Dazhi Huang, and H.P. In. Situation-aware contract specification language for middleware for ubiquitous computing. Distributed Computing Systems, 2003. FTDCS 2003. Proceedings. The Ninth IEEE Workshop on Future Trends of, pages 93–99, 28-30 May 2003.
[10] L. Zeng, 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, 30(5):311–327, 2004.
[11] G. Canfora, M. Di Penta, R. Esposito, and M.L. Villani. An approach for QoS-aware service composition based on genetic algorithms. Proceedings of the 2005 conference on Genetic and evolutionary computation, pages 1069–1075, 2005.
[12] A. Solberg, S. Amundsen, J.Ø. Aagedal, and F. Eliassen. A Framework for QoS-Aware Service Composition. Proceedings of 2nd ACM International Conference on Service Oriented Computing, 2004.
[13] Y. Tokairin, K. Yamanaka, H. Takahashi, T. Suganuma, and N. Shiratori. An effective QoS control scheme for ubiquitous services based on context information management. cec-eee, 00:619–625, 2007.
[14] Svend Frolund and Jari Koistinen. Quality of services specification in distributed object systems design. In COOTS'98: Proceedings of the 4th USENIX Conference on Object-Oriented Technologies and Systems (COOTS), pages 1–1, Berkeley, CA, USA, 1998. USENIX Assoc.
[15] H. Ludwig, A. Keller, A. Dan, R.P. King, and R. Franck. Web Service Level Agreement (WSLA) Language Specification. IBM Corporation, 2002.
[16] V. Tosic, K. Patel, and B. Pagurek. WSOL-Web Service Offerings Language. Web Services, E-Business, and the Semantic Web: CAiSE 2002 International Workshop, WES 2002, Toronto, Canada, May 27-28, 2002: Revised Papers, 2002.

[17] D.D. Lamanna, J. Skene, and W. Emmerich. SLAng: A Language for Defining Service Level Agreements. Proc. of the 9th IEEE Workshop on Future Trends in Distributed Computing Systems-FTDCS, pages 100–106, 2003.
[18] A. Andrieux, K. Czajkowski, A. Dan, K. Keahey, H. Ludwig, J. Pruyne, J. Rofrano, S. Tuecke, and M. Xu. Web Services Agreement Specification (WS-Agreement). Global Grid Forum GRAAP-WG, Draft, August, 2004.
[19] A. Polze and L. Sha. Composite Objects: Real-Time Programming with CORBA. In Proceedings of 24th Euromicro Conference, Network Computing Workshop, Vol. II, pp.: 997-1004, Vaesteras, Sweden, August 1998.
[20] W. Feng. Dynamic client-side scheduling in a real-time corba system. In COMPSAC, pages 332–333. IEEE Computer Society, 1999.
[21] Pascal Felber, Rachid Guerraoui, and André Schiper. Replication of corba objects. In Sacha Krakowiak and Santosh K. Shrivastava, editors, Advances in Distributed Systems, volume 1752 of Lecture Notes in Computer Science, pages 254–276. Springer, 1999.
[22] V. Marangozova and D. Hagimont. An infrastructure for corba component replication. In Judith M. Bishop, editor, Component Deployment, volume 2370 of Lecture Notes in Computer Science, pages 222–232. Springer, 2002.
[23] M. Werner. Replikation in CORE. Bericht an das Graduiertenkolleg "Kommunikationsbasierte Systeme", Oct 1996.
[24] Pascal Felber and Priya Narasimhan. Reconciling replication and transactions for the end-to-end reliability of corba applications. In Meersman and Tari [70], pages 737–754.
[25] Pierre-Charles David and Thomas Ledoux. An infrastructure for adaptable middleware. In Meersman and Tari [70], pages 773–790.
[26] Sebastian Gutierrez-Nolasco and Nalini Venkatasubramanian. A reflective middleware framework for communication in dynamic environments. In Meersman and Tari [70], pages 791–808.
[27] Gregory Biegel, Vinny Cahill, and Mads Haahr. A dynamic proxy based architecture to support distributed java objects in a mobile environment. In Meersman and Tari [70], pages 809–826.
[28] Sandeep Adwankar. Mobile corba. In DOA '01: Proceedings of the Third International Symposium on Distributed Objects and Applications, page 52, Los Alamitos, CA, USA, 2001. IEEE Computer Society.
[29] O. Babaoglu, A. Bartoli, V. Maverick, S. Patarin, J. Vuckovic, and H. Wu. A Framework for Prototyping J2EE Replication Algorithms.
[30] Etienne Antoniutti Di Muro. A software architecture for translucent replication. In DSM '05: Proceedings of the 2nd international doctoral symposium on Middleware, pages 1–5, New York, NY, USA, 2005. ACM.
[31] Lei Gao, Mike Dahlin, Amol Nayate, Jiandan Zheng, and Arun Iyengar. Application specific data replication for edge services. In WWW '03: Proceedings of the 12th international conference on World Wide Web, pages 449–460, New York, NY, USA, 2003. ACM.
[32] Michael M. Swift, Brian N. Bershad, and Henry M. Levy. Improving the reliability of commodity operating systems. ACM Trans. Comput. Syst., 23(1):77–110, 2005.
[33] Armando Fox, Steven D. Gribble, Yatin Chawathe, Eric A. Brewer, and Paul Gauthier. Cluster-based scalable network services. In SOSP '97: Proceedings of the sixteenth ACM symposium on Operating systems principles, pages 78–91, New York, NY, USA, 1997. ACM.
[34] Haifeng Yu and Amin Vahdat. The costs and limits of availability for replicated services. ACM Trans. Comput. Syst., 24(1):70–113, 2006.
[35] A. Rosenthal. Computing the Reliability of Complex Networks. SIAM Journal on Applied Mathematics, 32(2):384–393, 1977.
[36] Haifeng Yu and Phillip B. Gibbons. Optimal inter-object correlation when replicating for availability. In PODC '07: Proceedings of the twenty-sixth annual ACM symposium on Principles of distributed computing, pages 254–263, New York, NY, USA, 2007. ACM.

[37] Peter J. Denning. Acm president's letter: What is experimental computer science? Commun. ACM, 23(10):543–544, 1980.
[38] M.J. Norušis and S. Inc. SPSS 11.0 Guide to Data Analysis. Prentice Hall, 2002.
[39] B.D. Ripley. The R project in statistical computing. MSOR Connections. The newsletter of the LTSN Maths, Stats & OR Network, 1(1):23–25, 2001.
[40] Ann T. Tai, William H. Sanders, Leon Alkalai, Savio N. Chau, and Kam S. Tso. Performability analysis of guarded-operation duration: a translation approach for reward model solutions. Perform. Eval., 56(1-4):249–276, 2004.
[41] Krishna R. Pattipati and Samir A. Shah. On the computational aspects of performability models of fault-tolerant computer systems. IEEE Trans. Computers, 39(6):832–836, 1990.
[42] Gianfranco Ciardo, Raymond A. Marie, Bruno Sericola, and Kishor S. Trivedi. Performabilty analysis using semi-markov reward processes. IEEE Trans. Computers, 39(10):1251–1264, 1990.
[43] Kishor S. Trivedi, Antonio Puliafito, and Dimitris Logothetis. From stochastic petri nets to markov regenerative stochastic petri nets. In Patrick W. Dowd and Erol Gelenbe, editors, MASCOTS, pages 194–198. IEEE Computer Society, 1995.
[44] Bhuvan Urgaonkar, Giovanni Pacifici, Prashant Shenoy, Mike Spreitzer, and Asser Tantawi. An analytical model for multi-tier internet services and its applications. In SIGMETRICS '05: Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages 291–302, New York, NY, USA, 2005. ACM.
[45] S. Kounev and A. Buchmann. Performance Modeling and Evaluation of Large-Scale J2EE Applications. Proc. of the 29th International Conference of the Computer Measurement Group (CMG) on Resource Management and Performance Evaluation of Enterprise Computing Systems-CMG2003, 2003.
[46] Mohamed N. Bennani and Daniel A. Menascé. Resource allocation for autonomic data centers using analytic performance models. In ICAC '05: Proceedings of the Second International Conference on Autonomic Computing, pages 229–240, Washington, DC, USA, 2005. IEEE Computer Society.
[47] Daniel A. Menascé, Larry W. Dowdy, and Virgílio A.F. Almeida. Performance by Design: Computer Capacity Planning By Example. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 2004.
[48] Daniel A. Menascé, Virgílio A. F. Almeida, Rudolf Riedi, Flávia Ribeiro, Rodrigo Fonseca, and Jr. Wagner Meira. In search of invariants for e-business workloads. In EC '00: Proceedings of the 2nd ACM conference on Electronic commerce, pages 56–65, New York, NY, USA, 2000. ACM.
[49] Daniel A. Menascé and Virgílio A. F. Almeida. Scaling for e-business. Prentice Hall PTR Upper Saddle River, NJ, 2000.
[50] Daniel A. Menascé, Virgílio A.F. Almeida, and Larry W. Dowdy. Capacity Planning and Performance Modeling: From Mainframes to Client-Server Systems.Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1999.
[51] Daniel Villela, Prashant Pradhan, and Dan Rubenstein. Provisioning servers in the application tier for e-commerce systems. ACM Transactions on Internet Technology (TOIT), 7(1):7, 2007.
[52] S. Ranjan, J. Rolia, H. Fu, and E. Knightly. QoS-driven server migration for Internet data centers. Quality of Service, 2002. Tenth IEEE International Workshop on, pages 3–12, 2002.
[53] A. Kamra, V. Misra, and EM Nahum. Yaksha: a self-tuning controller for managing the performance of 3-tiered Web sites. Quality of Service, 2004. IWQOS 2004. Twelfth IEEE International Workshop on, pages 47–56, 2004.
[54] J.A. Rolia, K.C. Sevcik, et al. The Method of Layers. IEEE Transactions on Software Engineering, 21(8):689–700, 1995.
[55] E.D. Lazowska, J. Zahorjan, G.S. Graham, and K.C. Sevcik. Quantitative system performance: computer system analysis using queueing network models. Prentice-Hall, Inc. Upper Saddle River, NJ, USA, 1984.

[56] C.M. Woodside and G. Raghunath. General Bypass Architecture for High-Performance Distributed Applications. Proceedings of the Sixth IFIP WG6. 3 Conference on Performance of Computer Networks: Data Communications and their Performance, pages 51–65, 1996.
[57] Roy Gregory Franks. Performance analysis of distributed server systems. PhD thesis, Ottawa, Ont., Canada, Canada, 2000. Adviser-C. Murray Woodside.
[58] J. Xu, A. Oufimtsev, M. Woodside, and L. Murphy. Performance modeling and prediction of enterprise JavaBeans with layered queuing network templates. ACM SIGSOFT Software Engineering Notes, 31(2), 2005.
[59] Louis P. Slothouber. A model of web server performance. In Proceedings of the Fifth International World Wide Web Conference, 1996.
[60] R. Doyle, J. Chase, O. Asad, W. Jin, and A. Vahdat. Model-Based Resource Provisioning in a Web Service Utility. Proc. of the 4th USENIX Symp. on Internet Technologies and Systems.
[61] Abhishek Chandra, Weibo Gong, and Prashant Shenoy. Dynamic resource allocation for shared data centers using online measurements. In SIGMETRICS ’03: Proceedings of the 2003 ACM SIGMETRICS international conference on Measurement and modeling of computer systems, pages 300–301, New York, NY, USA, 2003. ACM.
[62] A. Chandra, P. Goyal, and P. Shenoy. Quantifying the Benefits of Resource Multiplexing in On-Demand Data Centers. Proceedings of the First Workshop on Algorithms and Architectures for Self-Managing Systems, 2003.
[63] B. Urgaonkar and P. Shenoy. Cataclysm: Handling Extreme Overloads in Internet Services. Proceedings of the 23rd Annual ACM SIGACT-SIGOPS Symposium on Principles of Distributed Computing (PODC), 2004.
[64] R. Levy, J. Nagarajarao, G. Pacifici, A. Spreitzer, A. Tantawi, and A. Youssef. Performance management for cluster based Web services. Integrated Network Management, IFIP/IEEE Eighth International Symposium on, pages 247–261, 2003.
[65] Tarek F. Abdelzaher, Kang G. Shin, and Nina Bhatti. Performance guarantees for web server end-systems: A control-theoretical approach. IEEE Transactions on Parallel and Distributed Systems, 13(1):80–96, 2002.
[66] Daniel A. Menascé. Web server software architectures. Internet Computing, IEEE, 7(6):78–81, 2003.
[67] G. Bolch, S. Greiner, H. de Meer, and K.S. Trivedi. Queueing networks and Markov chains: modeling and performance evaluation with computer science applications. Wiley-Interscience New York, NY, USA, 1998.
[68] Vladimir Stantchev, Trung Dang Hoang, Tino Schulz, and Ilja Ratchinski. Optimizing clinical processes with position-sensing. IT Professional, 10(2):31–37, 2008.
[69] Vladimir Stantchev and Christian Schröpfer. Techniques for service level enforcement in web-services based systems. In The 10th International Conference on Information Integration and Web-based Applications and Services (iiWAS2008),New York, NY, USA, 11 2008. ACM.
[70] Robert Meersman and Zahir Tari, editors. On the Move to Meaningful Internet Systems, Confederated International Conferences DOA, CoopIS and ODBASE 2002, Irvine, California, USA, Proceedings, volume 2519 of Lecture Notes in Computer Science. Springer, 2002.

Chapter 2

Trustworthiness Assessment Framework for Net-Centric Systems

Raymond Paul, Jing Dong*, I-Ling Yen*, and Farokh Bastani*,**

*University of Texas at Dallas, Richardson, Texas, USA

**Department of Defense, USA

Abstract Modern applications are becoming increasingly large-scale and network-centric, involving a variety of different types of system entities. Also, the assurance requirements for these systems are evolving due to the continuing emergence of new threats from new operational environments. To assure the trustworthiness of these systems to a sufficiently high degree of confidence is a challenging task. Most existing methods require different specialized assessment techniques for not only different types of system entities but also different trustworthiness aspects. Also, most existing techniques lack consideration of the overall system trustworthiness assessment from an integrated system perspective or fail to provide a holistic view. To address these problems, we develop an ontology-based approach to provide systematic guidelines for net-centric system assessment. The ontology-based approach captures evolving system trustworthiness aspects and effectively models their relationships and correlations. It can also organize system entities and associate appropriate assessment techniques for each class of system entities and their integrations.

1. Introduction

Due to the advances of computer and networking technologies, many applications are becoming large-scale and network-centric. A net-centric system (NCS) typically involves a distributed set of sensors, actuators, processors, software, along with a variety of other resources interconnected together by a network and interacting with and controlled by end users. Operational scenarios range from telecontrol and tele-monitoring systems to distributed coordination and communication systems, command and control systems, emergency response, and other areas.