

RTSOA: Real-Time Service-Oriented Architecture

W.T. Tsai, Yann-Hang Lee, Zhibin Cao, Yinong Chen, Bingnan Xiao
Computer Science and Engineering Department
Arizona State University,
Tempe, AZ 85287-8809, USA

Abstract

This paper extends the traditional Service-Oriented Architecture (SOA) to a new Real-Time SOA (RTSOA) and proposes a framework for the new architecture, by providing real-time service modeling, design, code generation, simulation, deployment, execution, orchestration, and management. The concepts, architecture, and enabling technique are studied for Real-time SOA. In addition, an efficient algorithm is derived in the paper to find the optimal composition of real-time services subject to the real-time constraint.

1. Introduction

As SOA penetrates into more and more application areas, applying real-time technologies to SOA becomes a necessity. Large companies such as CISCO, IBM, Microsoft, and TIBCO are pushing the technologies to enable real-time processing in SOA frameworks. Given that the characteristics of Real-Time SOA (RTSOA) are different from the current SOA, it is necessary to establish real-time support in SOA frameworks. This includes the approaches to meet timing constraints and to be predictable in many SOA aspects of modeling, composition, orchestration, deployment, policy enforcement, and management.

Real-time issues have been discussed in a number of studies. Kazi discussed the agile Real-Time Enterprise Architecture in [14], which is a combination of SOA and EDA for real-time business. Bodhare discussed service infrastructure optimization for real-time SOA, including redundancy, service scheduler engine, and service auditing in [2]. Most of these discussions focused on the quality of service enabled from the service provider's point of view, but do not address the overall issues of service integration and interoperability from the service consumer's or application builder's point of view.

This paper discusses the critical issues to enable RTSOA from both the service providers' and service consumers' sides. On the service providers' side, RTSOA provides the mechanism for real-time scheduling, resource management, and processing. On the service consumers' side, it provides support for real-time service modeling, application composition,

planning, service deployment, service execution, scheduling, orchestration, and management.

The rest of this paper is organized as follows. Section 2 discusses the real-time issues in SOA and the overall design of the RTSOA. Section 3 elaborates the major components of RTSOA frameworks. Section 4 presents a heuristic search algorithm for the optimal service composition. Comparisons between the proposed heuristic and exhaustive search algorithms are given in Section 5. Section 6 concludes the paper and discusses future work.

2. Real-time Perspectives in SOA

In RTSOA, service providers can enable real-time services, i.e. invocations of services must be completed within specific timing constraints. As well, the operations facilitated by SOA framework can follow real-time requirements. One of the examples is that the operation of composing services into SOA applications has bounded response time and resource usage.

For each SOA application, its lifecycle consists of a feedback loop of modeling, assembling, deployment, and management [11], as shown in Figure 1. The lifecycle of an RTSOA application follows a similar cycle. Thus, it is necessary to examine the support to common SOA techniques such as discovery, composition, analysis, simulation, deployment, orchestration, and management with respect to the needs of real-time computing.

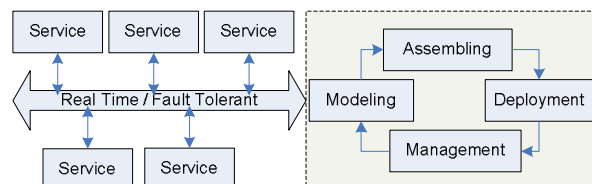


Figure 1: Real-Time SOA

Modeling and Specification: In this phase, individual services are specified, including functional and real-time properties. The workflow for the application is also depicted in this phase. Note that real-time properties should include the performance and timing requirement as well as workload characteristics, such as the service deadlines and invocation frequency.

Discovery and Assembly: Once the services are modeled and specified, the services required by the application can be discovered and assembled into the application. In this phase, control logic code is also generated to dynamically reconfigure and recompose the application. After the application is composed, multiple analyses including completeness & consistency checking, model checking (verification), simulation and testing (validation) can be performed to evaluate the system behaviors base on the application model and to analyze the real-time properties of the composed application. The steps of discovery, assembly, and validation should have bounded response times such as ESB (Enterprise Service Bus) []

Deployment: After the application is assembled and the composite service model evaluation results satisfy the requirements, the composed application can be deployed for execution. To permit real-time response for the application, the deployment step should make reservation of required resources at service provider and consumer sites, as well as interconnection networks.

Execution and Management: During the execution time, predetermined points of interests are monitored, policies are enforced, and data are collected, so that the system performance and behavior of the application can be evaluated. The service execution and monitoring should be scheduled and the reserved resource (CPU time, memory region, and network bandwidth) must be guaranteed. If the performance is not satisfactory, or faults are detected, reconfiguration or recomposition can be performed by changing by rebinding to better services or refining the composition specification. The recomposed application needs to be verified and validated.

Conceptually, services are connected to an abstract bus interconnection, such as ESB [11], which is independent of network routing and topology. Via the bus, services can be discovered, connected, and mediated. For RTSOA, the abstract bus is not only for interconnectivity and brokering, but also is with quality of service parameters for reliable real-time communication.

The RTSOA can be further enhanced by adding an SOA back-end to the architecture. In addition to provide support for real-time services, the new RTSOA framework needs to provide support for discovery and dynamic composition, deployment, and runtime monitoring in real-time. The real-time discovery and composition can be realized if service information can be cached by the SOA back-end and be retrieved within bounded search time. Similarly, with an assigned CPU and network bandwidth, the SOA back-end engine can carry the operations of SOA framework. For instance, the service verification and validation

may be done by the SOA back-end using the allocated CPU cycles and network bandwidth. As a consequence, verified services can be saved in a service repository to facilitate the service discovery for RTSOA.

3. Components of RTSOA Framework

To implement RTSOA, almost every techniques and protocols in traditional SOA need to consider the possible implications to real-time constraints. This section enumerates several key components in a RTSOA framework.

3.1. Real-Time Communication

Since the communication between services may happen between remote nodes in SOA application, real-time communication is a basic issue that needs to be resolved in RTSOA. To achieve real-time communication, messages to be exchanged in the service communication need to be serialized in real-time for marshalling/unmarshalling; Channel bandwidth between the services need to be reserved; Fault handling in application composition and communication backbone need to employ forward fault recovery;

Typically, the communication channels between service providers and consumers are formed with multiple layers of protocols. Services and applications first wrap application data in XML. Then, a message protocol, SOAP, is utilized to encode data in XML-based model. At transport layer, HTTP and encrypted HTTPS are commonly used over TCP/IP. For TCP connections, the two Internet QoS models, *integrated services* and *differentiated services*, can be invoked if there is a proper QoS routing support in the network.

In addition to the QoS at transport layer, the SOAP protocol processing may have a notable performance deficiency due to the slow and memory intensive message encoding and serialization [16]. It will be required to streamline the SOAP protocol processing, such as using a table-based serialization approach [10], the code-generation approach for XML serialization in gSOAP [7].

3.2. Service Modeling for Real-Time Properties

In additional to the functional service modeling, the real-time properties of services also need to be included in the service specification. A service may provide various levels of service quality and, at each service level, service specification may need to include the following information:

- Minimum and maximal response times
- Service capacity (such as a number of service invocations that can be accepted per unit of time)

- Degree of concurrency (the maximal number of service consumers that the service provider can be bounded to simultaneously)
- Cost and required resource

Service specification represents the service quality that can be guaranteed by the provider under the existing resource reservation. This implies that the service specification on real-time properties will be adjusted dynamically when additional resource is granted or commitments are made to service consumers.

3.3. Repositories for Real-Time Composition

For dynamic and real-time composition in RTSOA, advanced service discovery is necessary and the discovered services are stored in a repository that can be accessed in real-time. A three layered repository structure is used in RTSOA, as shown in the Figure 2, where, the repositories contain the links to pre-verified services and their real-time service property. Those verified services are ready to be used in a real-time application without additional testing. At runtime, the active services bound into the application will be stored in the service cache, the hot backup services are stored in memory, and the cold backup services are stored in remote service repository.

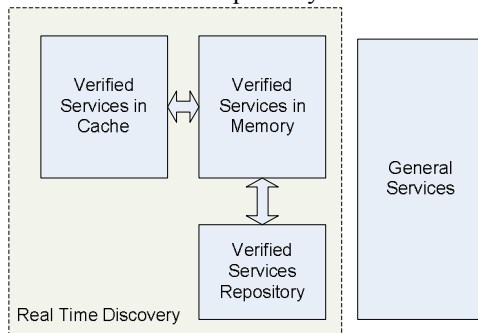


Figure 2: Three-Level Real-Time Discovery

3.4. Dynamic Service Composition

In RTSOA, applications can be dynamically composed in real-time. The service requirement is specified in the workflow of the composite service (application), as shown in Figure 3. Services to be included in the composite service (application) are decided based on the service ontology. In RTSOA, verified services are stored in the service cache or memory, and thus can be accessed rapidly according to the service requirements. In the service composition, different services will be selected at the run time. Then, the application will be ready for deployment.

The service composition operation is built on an optimization process in selecting the suitable services and to decide the quality level the application needs

from each services it invokes. For example, assume a workflow specification for an application is composed by a sequence of services:

$$A \rightarrow B \rightarrow C \rightarrow D$$

The top level application specification may include an end-to-end timing requirement for the entire workflow, for example, 10 seconds from *A* to *D*. The expected requests to the services are 100 requests/seconds. At the lower level services, timing properties may be specified as services *A* and *B* require 2 seconds to finish and services *C* and *D* requires 3 second to finish. In Section 4, an optimization problem is formulated and solution algorithms are discussed.

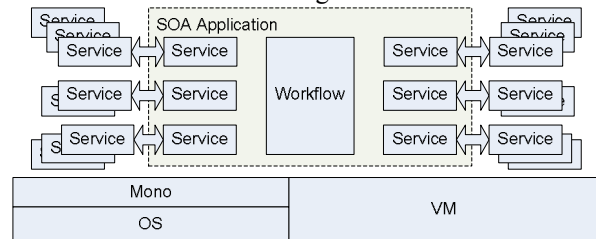


Figure 3 Real-Time Composition

3.5. Real-Time Service Deployment

In RTSOA, the composed service needs to be deployed for execution and the deployment operation must be done in real-time. Generally, there is an application host in the SOA framework for hosting the composed application. The simple control code and workflow specification will be deployed to the application host. Furthermore, there need to be an orchestration engine and data collection engine on the application host. To achieve real-time deployment, reserved bandwidth is necessary to ensure the data transfer can be finished in real time. Also, the data volume for deployment needs to be small so that the time for deployment can be reduced. Minimum installation or setup on the application host is also necessary to optimize the deployment.

The deployed application may consist of the serviced selected in the composition step. A binding process must be carried out between the application host and the service providers to ensure the reservation of resource and the guaranteed quality of service.

3.6. Data Collection and Policy Enforcement

At the application run time, data need to be collected for different analyses and policy enforcement. A real-time policy engine will be deployed in the RTSOA framework. Application behaviors need to be checked against the policies specified in real time. These policies include functional policies as system behavioral constraints and non-

functional policies including timing policies. If a policy is violated, the policy engine needs to execute the forward compensation operations for recovery.

3.7. Real-time Service Execution Environment

In the RTSOA framework, the execution of service providers and consumers are governed by the underlined execution environment, such as operating systems and virtual machines. In addition, competing execution entities may share system resources that are managed by the execution environment. Hence, a proper support to ensure QoS of services and applications must be provided by the environment.

In our RTSOA framework, we focus on VM-based execution environment, such as JVM (Java Virtual Machine) and Microsoft CLI (Common Language Infrastructure). They have been adapted widely in the existing SOA applications and enable an abstract computing environment for software portability. In addition, programs developed in Java or CLI-compatible languages are claimed secure due to type-safety and security sandbox model. The ensured safety, portability, and reusability of OO languages make VMs attractive to SOA and the integration of the two approaches can complement with each other in terms of software portability and interoperability.

Both JVM and CLI can be viewed as layered software libraries to provide system and language support operations, such as thread management, garbage collection, dynamic class loading, namespace, JIT (Just-in-Time compilation), AOT (Ahead-of-Time compilation), serialization, and programming language interface (e.g. JNI). To lead to QoS guarantees for RTSOA, a deterministic behavior in VM must be ensured, i.e., these VM services must be bounded in their WCET (worst-case execution time). Furthermore, VM's operations should be preemptible and schedulable in the similar manner as application threads, and allow the pause time due to the operations to be controlled [1][9].

Scheduling of application services and VM operations should address both the timing requirements of real-time operations and the dynamic features of SOA. A hierarchical scheduling mechanism [15] can be adopted:

1. *At partition level:* The notion of partitioning is used to denote the ownership of system resource, including CPU time, memory, and network connection at each service node. It consists of spatial and temporal partitioning among services/applications and acts as a brick wall such that task execution in a partition cannot be affected by any requests in other partitions. Partition can be assigned to each application domain in CLI, or a namespace in JVM. The CPU time can be reserved

in the form of *partition cycle*, η , and *partition capacity* α , i.e. for each partition cycle, the server can execute the tasks in the partition for an interval at least $\alpha\eta$ where α is less than or equal to 1.

2. *At thread level:* Multiple threads of a partition share the system resource owned by the partition and interact with each other to complete service invocations and applications. Threads can be scheduled according to their priorities and run concurrently to handle multiple and overlapping requests.

An example of the hierarchical scheduling mechanism is to enable a service provider in a partition where a thread pool is initiated and consists of a fixed number of threads. With distinctive priorities, differentiated services can be connected to service consumers. At partition level, the service provider can be scheduled periodically with a fixed CPU budget or as a sporadic server which is with constant bandwidth and replenishment period. Similarly, one or more partitions can be scheduled for VM operations, including garbage collection, class loading and JIT, etc, as shown in Figure 4.

3.8. Mechanisms for Real-time Guarantee

Real-time mechanisms for both the service providers and service consumers (applications) need to be designed to guarantee operations to be executed in real time. At the service provider side, issues in the real-time mechanisms include:

- Message queue design
- Message priority
- Operation preemption
- Multi thread scheduling

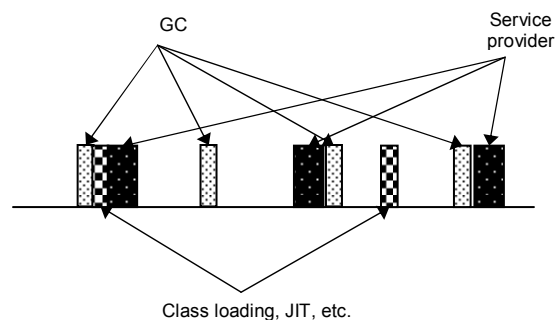


Figure 4: Partition scheduled for VM operation

In addition to a first-come-first-serve queue for incoming service request messages, the RTSOA framework also needs to prioritize the message queue according to different criteria including timing priorities.

Different service consumers may subscribe to different level of services that guarantee different levels of quality of services (QoS). Consumers with a high subscription rank can be assigned with higher priority than consumers with a low rank. The scheduling mechanism need to take care of the scheduling based on subscription level and timing priority. In addition, CPU execution budget and network bandwidth must be reserved for differentiated service. Overrun management can be installed to ensure the proper sharing of system resource.

When composing a new real-time application in RTSOA, even if the verified services are selected based on the service requirement specified in the application, the services may not collaborate with each other correctly because of timing compatibility. Real-

time V&V is necessary to verify that the services selected are compatible with respect to the timing constraints.

In the application workflow specification, timing constraints will be specified for all operations in the services. The task scheduling and planning mechanisms need to coordinate the collaborations among the services to guarantee that the timing constraints can be satisfied based on the service QoS levels that this application has subscribed to.

RTSOA is different from current SOA. The real-time requirements for RTSOA bring unique characteristics to the RTSOA framework. The characteristics of RTSOA and the comparison with current SOA is shown in Table 1.

Table 1: Comparison of Current SOA and RTSOA

		Current SOA	RTSOA
Basic SOA	Ontology	Service ontology and collaboration ontology is needed for service composition and collaboration.	Ontology specification and matching will be done offline to avoid the real-time issues. In addition to the general service information, real-time properties of services need to be included in the service ontology and collaboration ontology
	Discovery	UDDI and ebXML as registry for service discovery	Instead of the general service discovery in the current SOA, RTSOA can discover and choose only those verified services in the database to ensure real-time performance.
	Message Exchange	SOAP as the message exchanging protocol	Real-time SOAP or similar protocols are needed for real-time message exchange
	Deployment	General service deployment after the application is composed or recomposed	Real-time deployment mechanism is needed for deploying dynamically composed applications to the application host. Reserved deployment bandwidth is necessary for timing guarantee.
SOA Management	Orchestration	The service collaboration needs to be orchestrated at runtime	Real-time service orchestration is needed
	Policy	Policies need to be enforced at the service runtime.	Policy enforcement needs to be carried out in real time and at runtime. Real-time system recovery on policy violation.
Service-Oriented System Engineering	Modeling	Service modeling for general service information	In addition to the general service modeling, real-time service properties also need to be modeled.
	Composition	Service composition based on service ontology	Real-time service composition is required. It requires verified services to be available.
	V & V	Service verification and validation are needed before services are deployed.	Verification and validation are done offline before adding the services into the verified service repository. RTV&V may be needed in the case when verified services are not available. RTV&V is also used for service timing compatibility verification.
	Simulation	Both online and offline simulations can be performed for service behaviors evaluation.	Verified service simulation is done offline before adding the services into the verified service repository. Real-time online simulation mechanism is needed for dynamically composed application evaluation.
	Execution	After a composite service is composed and evaluated, it can be put into execution	Real-time properties need to be enabled in the service execution. Execution data will be collected for timing policy enforcement.
	Reconfiguration	Service reconfiguration is needed when reliability or performance is not low.	Service reconfiguration needs to be carried out in real time to satisfy the timing constraints.

4. Real-time Composition and Optimization

4.1. Problem Definition

In SOA applications, the payment for services is based on the usage. The same service can be provided at different levels, in terms of response time and cost. Hence, to select services for application composition, there are two targets to optimize: the total cost and the end-to-end execution time. More concretely, one can represent the optimization problem based on the following constraints and assumptions:

- The application must complete one iteration in *MaxTime* and can use up to a *MaxCost* budget;
- The application will call n services in a sequence: S_1, S_2, \dots, S_n ;
- Each service can guarantee k levels of service quality. For level l service S_i , we use $C_i(l)$ and $T_i(l)$ to represent the cost and response time of service S_i at level l . We assume service levels are sorted in an ascending order of response time (as well as a descending order of cost). Thus, the level 1 has the shortest response time and the highest cost;
- Given a composition $\{l(i), i = 1, 2, \dots, n\}$ as the service level we choose for service S_i , where $1 \leq l(i) \leq k$, we define the end-to-end response time and total application cost as:

$$Sum(T) = T_1(l(1)) + T_2(l(2)) + \dots + T_n(l(n))$$
 and

$$Sum(C) = C_1(l(1)) + C_2(l(2)) + \dots + C_n(l(n))$$
 We can have two optimization targets based on the constraints and assumptions:
- Objective 1: Find the appropriate levels $l(i)$ for each service to get the minimal cost $Sum(C)$ in all combinations, under the constraint that the end-to-end response time is less than the *MaxTime*, i.e., $Sum(T) \leq MaxTime$;
- Objective 2: Find the appropriate levels $l(i)$ for each service to get the minimum execution time $Sum(T)$ in all combinations such that the total application cost is bounded to *MaxCost*, i.e., $Sum(C) \leq MaxCost$;

4.2. Algorithm Design

This section presents a greedy algorithm to illustrate how to get the optimized result. We use the objective 1 as the example in this section. Similar algorithms can be applied to meet the objective 2.

An exhaustive search can certainly find the optimal composition by calculating the time and cost for all possible combinations, i.e., for all service levels and for each service. Some

reduction can be easily added, for instance, when a composition of $\{l(i)\}$ cannot meet the end-to-end deadline constraint, there is no need to consider a different composition $\{l'(i)\}$ where $l'(i) \geq l(i)$ for all i . Since the number of combinations is k^n , the complexity of this algorithm is exponential.

Obviously, this algorithm cannot be used in the RTSOA. There exist algorithms to find the best result under certain conditions. For example, if the cost and response time functions are treated as continuous and convex, Lagrange multiplier can be introduced to find the minimal value of a convex function. For real-time composition, one may prefer a fast heuristic algorithm that can result in a suboptimal solution with a guaranteed computation time.

For each service S_i , we assume the levels of services are sorted by the response time in ascending order. A heuristic search can start from the solution of $l(i)=1$, i.e. all services are done with the best quality and the maximal cost. If the solution is feasible, we can reduce the cost by lowering the quality of a service subject to the end-to-end deadline constraint. The cost reduction process first selects the service S_j that gives the maximal marginal gain of

$$\frac{C_i(l(i)) - C_i(l(i)+1)}{T_i(l(i)+1) - T_i(l(i))}$$

for all $i = 1, 2, \dots, n$. At the same time, the change of service level of S_j from $l(j)$ to $l(j)+1$ should not increase the end-to-end response time beyond the required deadline. The process can be repeated until no such a service can be found. The algorithm is summarized as follows.

A heuristic search for optimal composition:

```

service set  $S = \{S_1, S_2, S_3, \dots, S_n\}$ ;
 $l(i) = 1, \Delta T_i = T_i(l(i)+1) - T_i(l(i))$ , and
 $\Delta C_i = C_i(l(i)) - C_i(l(i)+1)$ , for  $i = 1, 2, \dots, n$ ;
compute  $Sum(T) = T_1(l(1)) + T_2(l(2)) + \dots + T_n(l(n))$ ;
create a sorted list of  $S$  according to the response time
in ascending order;
while ( the sorted list is not empty ) {
  remove  $S_k$  which is with the maximal  $\Delta C / \Delta T$  from
  the list;
  if  $Sum(T) + \Delta T_k \leq MaxTime$  then {
     $l(k) = l(k) + 1$ ;
    recomputed  $\Delta C_k$  and  $\Delta T_k$ ;
     $Sum(T) = Sum(T) + \Delta T_k$ ;
    insert  $S_k$  back to the sorted list;
  }
}

```

The heuristic search algorithm can be implemented with a sort list of the marginal gains of all services. The time complexity of this algorithm is $O(k*n*log(n))$ where, $k*n$ is the maximal number of iterations of the reduction process and $O(log(n))$ complexity is required to maintain the sorted list in each iteration. The efficacy of the algorithm is further validated by the simulation in the next section.

5. Simulation

The optimization algorithms are implemented to evaluate the relationship between the response time and the cost. In this evaluation, it is assumed that five services are used in the composition. Each service has 10 levels of response time. The cost of each service depends on its response time. The shorter the response time, the higher the cost. A set of sample data is given in Table 2, in which two formulas are used to generate the data:

1. Cost = 100/(Response Time)
2. Cost = 100/(1+log(Response Time))

Table 2. Response time versus cost

C/T	S ₁	S ₂	S ₃	S ₄	S ₅
L1	100 / 1	100 / 1	100 / 1	100 / 1	100 / 1
L2	50 / 2	59 / 2	50 / 3	50 / 4	59 / 3
L3	33 / 3	47 / 3	33 / 5	33 / 7	47 / 5
L4	25 / 4	41 / 4	25 / 7	25 / 10	41 / 7
L5	20 / 5	38 / 5	20 / 9	20 / 13	38 / 9
L6	16 / 6	35 / 6	16 / 11	16 / 16	35 / 11
L7	14 / 7	33 / 7	14 / 13	14 / 19	33 / 13
L8	12 / 8	32 / 8	12 / 15	12 / 22	32 / 15
L9	11 / 9	31 / 9	11 / 17	11 / 25	31 / 17
L10	10 / 10	30 / 10	10 / 19	10 / 28	30 / 19

In the exhaustive algorithm, all possible combinations are evaluated to obtain the minimum cost, under the constraint that the total response time does not exceed the given max response time. In this example, we set the required max response time to 50. The exhaustive algorithm finds the minimum cost is 116. One of the compositions is $(S_{1,1(8)}, S_{2,1(7)}, S_{3,1(6)}, S_{4,1(5)}, S_{5,1(6)})$. The response time for this composition is exactly 50. It uses 89235.23 microseconds to find this optimal result.

On the other hand, we can fix the cost to compute the best response time. If we set the total cost allowed to 150, the same exhaustive algorithm can find the best response time for the same data, which is 34 seconds. The combination that generates the best response

time is $(S_{1,1(6)}, S_{2,1(4)}, S_{3,1(4)}, S_{3,1(4)}, S_{4,1(4)})$. This exhaustive algorithm uses 63040.87 microseconds to find this result.

The exhaustive algorithm can always find the optimal solution, but its time complexity is exponential. In the next experiment, we vary the service number from 2 to 8, and each service uses the same cost model. We select the simple one Cost = 100/rtime. The execution time of the algorithm for the exhaustive algorithm increases significantly, as shown in Table 3. By giving the same data samples to the heuristic algorithm, the simulation results are listed in Table 4. **Error! Reference source not found.** compares the response times of the exhaustive and heuristic algorithms. As can be seen, its performance does not change significantly when increasing the service number. The minimal costs found from both algorithms are exactly the same, but the combination results are different in some cases.

Table 3. Response time versus cost for the Exhaustive Algorithm

Execution Time (μsecond)	Service Number	Minimal Cost	Selection of Combination
201.48	2	20	10,10,
802.60	3	30	10,10,10,
8799.39	4	40	10, 10, 10, 10,
93184.90	5	50	10, 10, 10, 10, 10,
984984.25	6	70	8, 8, 8, 8, 8, 10,
10691487.68	7	96	6, 6, 6, 8, 8, 8, 8,
112725981.00	8	124	6, 6, 6, 6, 6, 6, 6, 8,

To fairly compare these two algorithms, we randomly generate 15 sets of data samples. These samples have different service numbers, and the cost model of each service is randomly picked up, as discussed in the previous section. Although the service combinations found are different, in all case the best results are found.

Table 4. Response time versus cost for the heuristic Algorithm

Execution Time (μsecond)	Service Number	Minimal Cost	Selection of Combination
45.25	2	20	10,10
72.57	3	30	10,10,10
128.18	4	40	10, 10, 10, 10
157.37	5	50	10, 10, 10, 10, 10
188.65	6	70	10, 8, 8, 8, 8, 8
203.51	7	96	8, 8, 8, 8, 6, 6, 6
224.46	8	124	8, 6, 6, 6, 6, 6, 6, 8

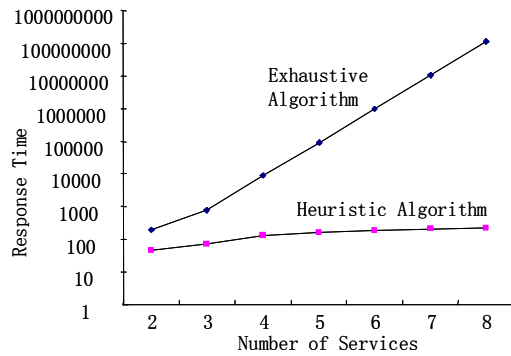


Figure 5. Execution time comparison

6. Summary

This paper presented a framework for RTSOA and various design considerations to achieve real-time communication, processing, and composition. The key issues include a pre-verified repository of services, reserved bandwidth, computing capacity, and trade-off between performance and cost. An efficient algorithm is developed to find the optimal cost for a given response time. The same algorithm can be used to find the best response time for a given cost. Its execution time and the capacity of finding the best solution are evaluated and compared with an exhaustive algorithm.

References

- [1] D. F. Bacon, P. Cheng, and V. T. Rajan. "A Real-Time Garbage Collector with Low Overhead and Consistent Utilization," *ACM SIGPLAN Notices*, 38(1):285-298, Jan. 2003.
- [2] S. Bodhare, "Optimizing Service Infrastructures", <http://blogs.ittoolbox.com/eai/optimization/archives/optimizing-service-infrastructures-3928>
- [3] D. Chappel, *Enterprise Service Bus: Theory in Practice*, O'reilly Media, 2004.
- [4] Y. Chen and W.T. Tsai, Introduction to Programming Languages: Programming in C, C++, Scheme, Prolog, C#, and SOA, second edition, Kendall/Hunt Publishing, 2006.
- [5] CISCO SYSTEMS, "CISCO's relationship with TIBCO makes the real-time business more visible and secure", http://www.cisco.com/application/pdf/en/us/guest/products/ps6438/c1625/cdcont_0900aecd802f5ebf.pdf
- [6] DARPA: OWL-S: www.daml.org/services/owl-s/
- [7] R. van Engelen, "An XML Web Services Development Environment for Embedded Devices," <http://www.cs.fsu.edu/~engelen/cases03.html>
- [8] X. Fu, T. Bultan, and J. Su, "Formal Verification of E-Services and Workflows," Proc. Workshop on Web Services, E-Business, and the Semantic Web, LNCS 2512, Springer-Verlag, 2002, pp. 188-202.
- [9] O. Goh, Y.-H. Lee, Z. Kaakani, and E. Rachlin.

"A Schedulable Garbage Collection for Embedded Applications in CLI," *The 11th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA05)*, July 2005.

[10] J. Helander and S. B. Sigurdsson, "Self-Tuning Planned Actions Time to Make Real-Time SOAP Real," *The 8th IEEE International Symposium on Object-Oriented Real-Time Distributed Computing (ISORC'05)*, pp. 80-89.

[11] IBM, "IBM SOA Foundation: An Architectural Introduction and Overview", <http://download.boulder.ibm.com/ibmdl/pub/software/dw/webservices/ws-soa-whitepaper.pdf>.

[12] Intel: Service-Oriented Enterprise, The Technology Path to Business Transformation. http://www.intel.com/business/bss/technologies/soe/soe_background.pdf

[13] N. Kavantzaz, D. Burdett, T. Fletcher, and Y. Lafon, "Web Services Choreography Description Language", Version 1.0, W3C Working Draft 17 Dec. 2004. <http://www.w3.org/TR/ws-cdl-10/>

[14] A. Kazi, "Enabling Real-Time Business Through Service-Oriented&Event-Driven Architecture", <http://www.bijonline.com/index.cfm?section=article&aid=19>

[15] D. Kim, Y. H. Lee, M. F. Younis, "SPIRIT-μKernel for Strongly Partitioned Real-Time Systems," *The 6th International Conference on Real-Time and Embedded Computing Systems and Applications (RTCSA 2000)*, pp. 73-80

[16] C. Kohlhoff and R. Steele, "Evaluating SOAP for High Performance Business Applications: Real-Time Trading Systems", Proc. of WWW'03, 2003.

[17] Frank Leymann, Web Services Flow Language, Version 1.0, May 2001. <http://www-306.ibm.com/software/solutions/webservices/pdf/WSFL.pdf>

[18] OASIS: ebXML: <http://www.ebxml.org/>

[19] OASIS: Business Process Execution Language for Web Services (BPEL4WS), 2003. <http://xml.coverpages.org/bpel4ws.html>

[20] R. Paul, "DoD Towards Software Services", Tenth IEEE International Workshop on Object-oriented Real-time Dependable Systems (WORDS 05), February 2005, pp. 3-6.

[21] M. P. Singh, M. N. Huhns, *Service-Oriented Computing*, John Wiley & Sons, 2005.

[22] W.T. Tsai, Ray A. Paul, Bingnan Xiao, Zhibin Cao, Yinong Chen, "PSML-S: A Process Specification and Modeling Language for Service Oriented Computing", *The 9th IASTED International Conference on Software Engineering and Applications (SEA)*, Phoenix, November 2005, pp. 160-167.

[23] W.T. Tsai, C. Fan, Y. Chen, and R. Paul, "DDSOS: A Dynamic Distributed Service-Oriented Simulation Framework", in *Proceedings of 39th Annual Simulation Symposium (ANSS)*, Huntsville, AL, April 2006, pp. 160-167.

[24] O. Zimmermann, S. Milinski, M. Craes, F. and Oellermann, "Second Generation Web Services-Oriented Architecture in Production in the Finance Industry", *OOPSLA'04*, Oct. Vancouver, 2004.