Service-oriented middleware: A survey

Jameela Al-Jaroodi*, Nader Mohamed1
Faculty of Information Technology, UAEU, PO Box 17551, Al Ain, United Arab Emirates

**Abstract**

Service-oriented computing aims to make services available and easily accessible through standardized models and protocols without having to worry about the underlying infrastructures, development models or implementation details. This helps achieve interoperability and loose coupling among distributed application components and also among user processes. In addition, this model offers users an on-demand usage model where they only use the services needed for the time needed, which relieves them from having to build and maintain a complete system in house. However, the design and implementation of robust and efficient service-oriented applications are still as complex and demanding as any other type of distributed application. Thus middleware can play an important role in facilitating the design, development and implementation of service-oriented systems. Furthermore, middleware approaches will provision non-functional requirements like performance, scalability, reliability, flexibility and quality of service (QoS) assurance. A lot of work has been done in this area and in this paper we survey some of this work in service-oriented middleware (SOM). As we study the different projects we develop a list of the main requirements that SOM should support. We also discuss the main objectives and characteristics of the surveyed approaches, and then we highlight the challenges to be addressed when designing and developing SOM solutions that satisfy the requirements of different application domains.

© 2011 Elsevier Ltd. All rights reserved.

**Contents**

1. Introduction .......................................................... 212
2. Background and concepts ........................................... 212
3. SOM requirements .................................................... 213
4. SOM approaches ..................................................... 214
   4.1. Cumulus .......................................................... 216
   4.2. Colombo .......................................................... 216
   4.3. SO-CAM .......................................................... 216
   4.4. ubiSOAP .......................................................... 216
   4.5. (SI)2 ............................................................... 216
   4.6. RESCUE .......................................................... 216
   4.7. OMG DDS with SIP ............................................. 217
   4.8. SOA-MM .......................................................... 217
   4.9. QoS SOM .......................................................... 217
   4.10. NINOS ........................................................... 217
   4.11. B-VIS ............................................................. 217
   4.12. SAI ............................................................... 217
   4.13. Weka4WS ....................................................... 217
   4.14. CROWN-C ....................................................... 218

*Corresponding author. Tel.: +971 3 7135518; fax: +971 3 767 2018.
E-mail addresses: jaljaroodi@gmail.com (J. Al-Jaroodi), nader.m@uaeu.ac.ae (N. Mohamed).

© 2011 Elsevier Ltd. All rights reserved.
1. Introduction

Distributed applications gained a strong momentum through the advances of technology in distributed computing, networking, embedded systems, heterogeneous platforms, and sensor networks. These advances offered an exceptional opportunity for developing a wide array of distributed applications like Grid, cloud and ubiquitous applications, which are essential for the progress of different fields such as medicine, environment protection, manufacturing, engineering and businesses. In general, large scale and distributed applications are built utilizing different heterogeneous devices such as high-end servers, RFID components, mobile devices and storage systems. These devices communicate through a variety of wireless/wired networking technologies. In addition, these devices are usually controlled by a variety of software components developed by different vendors and implemented using different programming languages and models. Due to the high distribution, dynamic properties and heterogeneity of the devices, software components and communication technologies used, implementing such applications is a non-trivial task. There are usually integration, scalability, reliability, security, usability, QoS and operational issues to be considered.

As a result, the concept of offering tools and software components to assist in utilizing and integrating all these components in the form of services has evolved to help overcome some of the problems. This approach provides an opportunity to make many software components and functionalities available for users as a set of independent services to be integrated in their applications based on their specific needs rather than implementing full solutions from scratch. In addition, this also allows for better testing and standardization of such components since they will be offered for reuse and integration to various interested consumers.

Recently service-oriented computing (SOC) (Bichler and Lin, 2006) has become a preferred model over the more traditional monolithic and client/server development models. It overcomes several problems and challenges these two types have faced. However, SOC is not a perfect solution either. As in any other approach, SOC suffers from serious issues in supporting heterogeneous environments and systems and in providing effective functional and non-functional requirements for the applications. Thus middleware becomes an important approach to use for SOC.

As a result, the service-oriented middleware (SOM) has become very important.

Using SOM to support services development and utilization has been advocated by many researchers, who call for using service-oriented middleware (SOM) over traditional middleware that connects diverse components and systems, and provides multiple-channel access to services over a variety of networks, including the Internet, UMTS, XDSL and Bluetooth (Papazoglou et al., 2006). Researchers also found web services to be one of the best technologies for the implementation of SOM (Lee et al., 2005). One of the most important indicators of the popularity of SOM is the development of a reference model for B3G-oriented SOM (Autili et al., 2008). The group designed the model to identify the issues to consider while architecting middleware solutions for service-oriented applications over B3G (Www link, 2010). The aim is to establish a common understanding of the support that should be provided by a generic B3G-SOM. On another aspect, the authors (Maia et al., 2009) identify the requirements and challenges of SOM for pervasive environments. Among the requirements there are primary requirements such as message exchange, mobility support, security and service description, discovery and coordination. There are also cross-cutting requirements including QoS support, autonomy and context awareness. Furthermore, a survey on SOM for wireless sensor networks (WSN) identifies several approaches in that domain and presents several requirements and challenges for SOM to support WSN (Mohamed and Al-Jaroodi, 2011). The SOM design for WSN faces different challenges and also has different requirements imposed mainly by the limitations in the available resources in WSN and WSN devices. These limitations in power sources, processing capabilities and memory make it important to design the SOM under strict restrictions. For example, it has to have a very small footprint, minimum communications requirements and high reliability and fault-tolerance capabilities. In this survey we concentrate on the generic model serving high-end computing facilities that do not have such limitations. Therefore, the approaches, requirements and challenges differ significantly.

In this paper we review and discuss different approaches SOM, and present some of the challenges they face. Thus in Section 2 we provide some background information and related concepts. The SOM requirements are discussed in Section 3 and the current approaches are discussed in Section 4. Section 5 provides a discussion of the SOM challenges and Section 6 concludes the paper.

2. Background and concepts

Web services immerged earlier as an easy to use technology to support the concept of software as a service (SaaS). They provide developers with XML-based methods to access and integrate services available over the Internet and utilize them as part of their own applications. The concept further advanced with time to result in a larger, more sophisticated framework under service-oriented computing (SOC) or as some call it services computing (CS). The main advantage of this approach is giving the applications a way to integrate various services available online within the context of the application’s specific domain and using them as needed instead of implementing the whole solution from scratch.

Middleware also offers similar capabilities to integrate and reuse software components on demand. However, it does not support online or on-demand integration easily. It is generally more targeted for in-house solutions rather than service access. Some researchers view SOC as a replacement for middleware since application development will not rely on detailed implementations, but will mainly integrate with available components based on SOC. However, even the SOC architecture itself has grown drastically that it resembles very large scale applications and require careful and well planned design and development. As a result, middleware approaches are used to facilitate the design, development and integration of services. Middleware helps to abstract the distribution and heterogeneity of the underlying computing environment and services available. It also supports the addition of non-functional features such as performance, scalability, reliability, availability, usability, manageability, QoS, efficiency and security. A number of middleware platforms were developed to support values added systems in various domains such as enterprise systems (Britton, 2001), cluster computing (Al-Jaroodi et al., 2003), wireless sensor networks (WSN) (Hadim and Mohamed, 2006), mobile ad hoc networks (Hadim et al., 2006) and robotics (Mohamed et al., 2008). The mechanisms and
approaches are usually based on the reuse of existing methods, protocols and software to add needed values. This approach fits nicely within the concept of service-oriented computing. Furthermore, it has become important to model SOC and SOM architectures in a formal way to represent and analyze the functional and non-functional properties. These models could also be the basis to easily transform the design into an executable form. One example of this effort is found in Gilmore et al. (2010), where formal methods are used to model the non-functional properties of service-oriented systems.

In industry, SOC (and in a relatively similar context SOA (service-oriented architecture)) offer a promising way for enabling software vendors to present their software applications as services. It provides a framework to represent business processes as independent modules (services) with clear and accessible interfaces. The interactions among services are done through a standard description language such as XML, which makes it easy to integrate different services to build a business application and address problems related to the integration of heterogeneous applications in a distributed environment. Services can implement a single business process or a set of different processes that are made available for integration with other heterogeneous services. Services can be developed using a wide range of technologies, including SOAP, REST, RPC, DCOM, CORBA and Web Services. Each one of these approaches provides its own model for designing and implementing service-oriented applications; however, they all have similarities in the general model. The generalized model basically involves three main players: the service provider, the service broker and the service consumer. The service provider designs and develops a service. The service broker makes this service available to the rest of world through public registries such as Universal Description Discovery and Integration (UDDI) for web services. The service consumer locates the entries in the public registry and binds with the service provider to invoke the web services required. There are many people (researchers and practitioners) with many different levels of understandings and uses of service-oriented computing.

According to Papazoglou (2003), the SOC approach makes the following things possible:

- Multiple applications executing in varied platforms can effectively communicate with each other.
- Software services are provided to end-user applications through published/discoverable interfaces.
- Services are loosely coupled thus reducing dependencies and facilitating reuse.
- Service consumers are coupled with services needed for the duration of the utilization only.
- A relationship is created among participants: service provider, service broker and the service consumer.
- Interactions between the participants involve the publish, find and bind operations.
- Service consumers target specific services through specific interfaces that are exposed by the service registry and discovery mechanisms offered through the service brokers.

Furthermore, SOC is becoming more practically popular and is supported by various large IT organizations. Oracle Corporation provides SOA through their Oracle Fusion Middleware (Knufsend, 2006), where several SOA-based solutions are provided to integrate other tools and applications such as Microsoft products to achieve better interoperability among Oracle-based applications and other applications. Another example is the SAP NetWeaver (Silver, 2009), which provides business process management (BPM) holistic approach to business solutions. NetWeaver BPM provides flexible and efficient models to integrate business processes (as services) to support different application requirements. Progress Open Edge (2007) is another example of solutions adapting SOA to provide flexible and effective utilization of services to provide holistic business IT solutions. iWay (Senor, 2005) provides an SOA foundation to enable the integration of business components such as applications, databases, files, communication transports, and e-business exchanges. In Ibrahim and Le Mouel (2009) the authors propose a middleware solution to support the dynamic composition of services in highly heterogeneous and changing pervasive environments. Furthermore, the business driven cloud computing has strengthened the SOC approaches and increased the demand on services in the business sector.

On the other hand, the research community is also putting a lot of efforts to provide usable, efficient and flexible tools and approaches for SOC. Services computing and cloud computing research is intensifying and offering solutions and approaches to make SOC more usable and feature-rich. In general several areas are under investigation including web services, service-oriented computing, service-oriented middleware, SOA middleware and Integration software. All of which represent variations of the concept of facilitating integration and simplifying application development.

3. SOM requirements

To begin, it may be necessary to try and define service-oriented middleware (SOM). From the name, SOM is a middleware designed as a collection of services that can be used to facilitate the development, execution and management of service-oriented applications. SOM like the other types of middleware is based on the multi-tier architecture, where it would reside somewhere between the main players in SOC (i.e. service providers, service developers and service consumers). SOM is necessary to support several, otherwise hard to incorporate, functionalities in the SOC model. These functionalities include not only the functional requirements, but also the non-functional requirements that different services might need to offer. In many application domains, services provide a set of specific functions based on the required application logic, which will be different for each application. However, those services and service providers also need to support several common functionalities that are generally irrelevant to the main application. For example service-oriented systems need to support service registries and discovery mechanisms and they also need to provide some level of abstraction to hide the underlying environments and implementation details. Furthermore, all SOC applications need to support some acceptable levels of reliability, performance, security and QoS. All of these can be supported and made available through a common middleware platform instead of having to incorporate them into each and every service developed. In general, based on our observations SOC is mainly business driven and to succeed SOC applications need to support the various demands of business and enterprise domains. In general, we can sum up the main requirements in a comprehensive generic SOM to be the following:

1. Support for service providers to build their services in a standardized model. Thus the SOM should offer usable development APIs that will facilitate the development process and ensure compliance with SOC standards. As a result service developers will not need to be involved in too many of the standardization and compliance issues and focus on the actual features of the services. In addition, this will result in a more uniform set of services being developed even if they were not developed by the same group.
2. Runtime support for service providers to deploy services and advertise them and ensure their availability and reliability.
Developers create services for their own applications needs all the time; however, it is usually hard to make these available to others. Therefore, SOM should provide developers with easy to use tools that help in publishing available services to others and making these services available for reuse. In addition, the SOM should also offer some facilities to support the availability and continuity of the services to avoid problems with service consumers. One way to manage reliability is to have some way of predicting the reliability levels (Zheng and Lyu, 2010).

3. Support for service consumers to discover and effectively use published services. When application designers require specific services, it may be hard to find out if they exist. In addition when a needed service is found, it may not be easy to integrate it with the application being developed. Therefore, it is necessary that the SOM provides service discovery tools and an easy model to integrate and use the discovered services. In addition, the SOM should be able to handle the differences in service consumer needs and integration capabilities.

4. Abstractions to hide the heterogeneity of underlying environments. This is a very important issue since applications and services are implemented independently from each other and in so many different development and operating environments. In addition many services may be using different devices and hardware infrastructures. Therefore, simply complying with a certain standard for SOC, is not enough to ensure smooth integration and interoperability across services, applications and platforms. The SOM should provide supporting APIs and protocols that allow for seamless integration without considerations for the differences in environments and development tools.

5. Service integration transparency to client applications. Ideally, a client's application should not be aware of the underlying configuration and composition of the application in use. Services should be integrated with the application without affecting any operating factors the users deal with. In addition, services need to be properly configured to provide best possible operational conditions to the user. Therefore, from the client's perspective, everything is available as a single packaged solution that provides the user with the required functionalities. The SOM should provide enough support to configure and integrate services and hide the details form the client's applications.

6. Adaptive/autonomous service discovery and integration mechanisms. This is a more advanced feature that should be available to ensure continuity of services. An application using services, should be able to have continuous access to these services and in case any of them fails (due to server, network or hardware failures), the SOM should have an autonomic discovery mechanism to find a replacement service (from the same or other provider) and seamlessly binds it with the application. This requirement includes the ability to discover service failures, discover and assess available replacements and seamlessly binds the selected replacement with the application during runtime. However, in a worst case scenario, if a replacement service is not available the SOM should be able to record current status and inform the user application of the problem.

7. Capability to handle high access rates, large volumes of data and high communication loads. Many services are used extensively and have high access demands on them. In addition many services generate large volumes of data thus imposing high loads on the system and the communication infrastructure. The SOM should provide well designed mechanisms to support the development of efficient service and data management functions. This should include features like replicated service management, high volume data transfers, efficient storage mechanisms and fast network access modules.

8. Support for reliable and secure operations. Reliability and security increase applications availability. With SOC, it becomes more important to support these features because applications have to trust executable components from various remote providers, while being able to reliably access them any time. Here it is necessary to support reliability and security at two main levels: communications, where the networking infrastructure is the main target; and at the operational level, where applications need to securely and reliably find, integrate and use the required services. SOM can be of great benefit in this context and at both levels by providing well tested SOM modules supporting reliability and security in a generic form that may be incorporated in various types of service-oriented applications. SOM should provide service protection tools and maintain service level security and trust. As a result, application developers will be able to use these tools as an integral part of the applications and protect their application and the services bound to it. In this regard, some of the security issues for SOM were discussed in detail in Al-Jaroodi and Al-Dhaheri (2011).

9. Support for QoS requirements. Once more, this is important for just about any distributed application. Therefore, when designing SOM to support service-oriented applications, developers should be able to specify different QoS requirements to suit the application. The SOM should provide the necessary features as part of the APIs such that QoS policies can be easily defined and enforced in the application.

Based on the list of requirements, it is clear that the first three requirements are important for any SOM claiming to provide basic features to support SOC. They involve the development, publishing and usage of services, which are essential to any SOC application. The remaining six requirements offer a value-added SOM that developers may choose not to include in the SOM design. However, their availability will increase the effectiveness, efficiency, security and availability of the service-oriented applications developed and deployed using this SOM. Generally, researchers and practitioners propose SOM approaches best suited for their underlying environments and for the targeted applications. Therefore, many of these approaches may provide some of these requirements, but most of them do not support all the requirements.

4. SOM approaches

While conducting our literature search through the different approaches researchers and practitioners are working on, we came across a large pool of examples of SOM. Several approaches are currently being investigated at the research level while a few are actually being used in practice. For our survey purposes we selected some examples of these approaches that we thought represent the most common SOM approaches and functionalities. The SOM approaches cover a wide variety of application domains and usually each one is designed to serve a specific purpose. Consequently, each SOM has its own unique design and set of features that it offers. However, most of them share at least a few common requirements from the list defined in Section 3. The following is a brief overview of the approaches we selected and following that, in Section 5, we discuss some of the common features and challenges involved. In addition, Table 1 summarizes the projects we selected in terms of the environment they were designed for, the main features they offer, the types of
## Table 1
Summary of the SOM approaches and their main features.

<table>
<thead>
<tr>
<th>No.</th>
<th>SOM</th>
<th>Platform/Environment</th>
<th>Main Features</th>
<th>Application domain</th>
<th>Requirements (explicitly covered)</th>
</tr>
</thead>
</table>
| 1   | Cumulus | General heterogeneous distributed environments | ● Runtime web service interoperability  
     |       |                      | ● Policy management  
     |       |                      | ● Uses BPEL and WSDL  | General | 1, 2, 3, 4 and 9 |
| 2   | Colombo | General distributed environments | ● Tools to develop, deploy and execute services  
     |       |                      | ● QoS support  
     |       |                      | ● Services are language independent | General | 1, 2, 3, 8 and 9 |
| 3   | SO-CAM | General distributed environments | ● Context awareness  
     |       |                      | ● Rules management and user notifications | General event-based | 2, 3 and 6 |
| 4   | ubiSOAP | Heterogeneous distributed environments, wired, wireless and mobile devices | ● Seamless networking of web services  
     |       |                      | ● Multi-radio networking and selection of best network for QoS requirements | General pervasive | 2, 4, 5, 7 and 10 |
| 5   | (SI)² | Heterogeneous distributed environments and smart items networks | ● Platform-independent service description and deployment  
     |       |                      | ● Support for smart items: sensors, RFID, and embedded devices | General and smart apps. | 1, 2, 3, and 4 |
| 6   | RESCUE | Heterogeneous distributed environments and mobile ad hoc networks | ● Effective mobile communication during critical missions  
     |       |                      | ● Reduced network traffic and fast change propagation  
     |       |                      | ● Real-time notifications of service availability | Ad hoc rescue applications | 2, 3, 4, 5 and 6 |
| 7   | OMG DDS | Heterogeneous distributed wired/wireless environments | ● Support real-time all IP communications  
     |       |                      | ● Supports QoS and adapts to changes  
     |       |                      | ● Provides universal access to information | Ad hoc rescue applications | 2, 3, 4, 6, 7 and 9 |
| 8   | SOA-MM | Distributed enterprise systems and shop floor equipment | ● Integrating shop floor operations with enterprise applications  
     |       |                      | ● Support for heterogeneous devices and networks | Business and manufacturing | 2, 3 and 4 |
| 9   | QoS SOM | General distributed environments | ● Support for complex inter-organizational interactions  
     |       |                      | ● Workflow management and QoS support | Business processes | 2, 3 and 9 |
| 10  | NINOS | General distributed environments | ● Offers distributed service orchestration for business processes  
     |       |                      | ● Uses fine-grained content-based routing and event correlation  
     |       |                      | ● Offers reliability through load balancing and fault tolerance | Business processes | 2, 3, 4, 8 and 7 |
| 11  | B-VIS | Heterogeneous distributed environments and networked RFID/Sensor devices | ● Real-time tracking and monitoring  
     |       |                      | ● Programming model to retrieve real-time data and integrates with enterprise systems | Business and Enterprise | 2, 3 and 4 |
| 12  | SAI | Heterogeneous distributed and ubiquitous environments | ● Context-aware service development  
     |       |                      | ● Integrates ubiquitous systems with enterprise applications | Business and Enterprise | 1, 2, 3, 5 and 7 |
| 13  | Weka4WS | Distributed Grid environment | ● Supports web service-based distributed data mining  
     |       |                      | ● Supports asynchronous invocations and results notifications  
     |       |                      | ● Separates computational services from client-side computations and interfaces | Distributed data mining | 2, 3, 4, 5 and 7 |
| 14  | CROWN-C | Distributed Grid environment | ● Supports fault tolerance and security features for grid services  
     |       |                      | ● Designed on top of CROWN and offers multiparty authentication and auto trust negotiation tools. | Grid applications | 2, 3 and 8 |

SOM requirements involve the support for:
1. Service composition and standardization
2. Service registry and publishing
3. Service discovery and integration
4. Heterogeneity
5. Integration transparency to client applications
6. Adaptation and Autonomicity
7. Scalability and efficiency
8. Reliability and security
9. QoS requirements.
4.1. Cumulus

Cumulus (Wohlstadter et al., 2006) is an SOM specifically designed for runtime web services interoperability. The architecture of Cumulus is comprised of the Cumulus Client, a set of remote middleware services and a middleware services registry. To implement the changes in the runtime interoperability requirements, the policy component within the cumulus client matches the client’s policy with the provider’s policy. Client policies can be attached to the business process execution language (BPEL) statements while the provider policies can be attached to the web services description languages (WSDL) definitions. These requirements help the framework discover the appropriate middleware service from the registry. After discovering the middleware service it composes the service and injects it into the process flow using the cumulus protocol. Cumulus offers a generic SOM model that can be used to build service-oriented applications in various domains.

4.2. Colombo

Colombo (Curbera et al., 2005) is a programming model plus a runtime support environment that provides SOC developers with the tools to develop, deploy and execute services. The Programming Model provides mechanisms to access and use other services, encode compositional logic, encapsulate the composition as a new service and state and honor QoS requirements. The unit of development and deployment in Colombo is a “servicelet,” which provides one or more services. Servicelets may be programmed in any language, invoke using a “stub” via an “adapter” and driven by messages. Developing a servicelet begins with a WSDL file from which mappings from XML schema data types to Java types, a skeleton Java code and QoS policies are generated. The runtime support contains three parts: SOAP-based message processing engine, Servicelet Managers and QoS Framework. It provides a built-in runtime support to deploy and execute the services created in Colombo.

4.3. SO-CAM

Service-oriented context-aware middleware (SO-CAM) (Santos et al., 2007) is a context-aware middleware which is the integration of two components: the context management service (CMS) (Ramparany et al., 2007), which supports context sources to publish their contextual information to be used by the context-aware applications; and the awareness and notification service (ANS) (Santos et al., 2007), which offers a rule-based facility allowing client applications to subscribe and specify rules containing context-based conditions and receive notifications when the specified contexts hold. The CMS architecture consists of three components: the context source (CS), which provides context information of a specific type to the CC (context consumer) either synchronously or asynchronously; the context broker (CB), which acts as a service directory for CS; and the CC, which uses the context information provided by CS and it also finds the appropriate CS by asking the CB directly. On the other hand, the ANS architecture consists of four main sub-components: the Event Monitor, the Controller, the Notifier and the Rule Manager. Context data events coming from CMS are received by the Event Monitor and get forwarded to the Controller which monitors them and evaluates the rules registered by the Rule Manager and stored in the Knowledge Repository. When the triggering condition of the rule becomes true, the Notifier is called to perform the suitable action. This middleware serves as a generic SOC middleware supporting generic event-based applications.

4.4. ubiSOAP

The main objective of designing ubiSOAP communication middleware (Caporuscio et al., 2008, 2011) is to provide seamless networking of web services that may be deployed on various devices including mobile devices. The architecture of ubiSOAP consists of a multi-radio networking layer and a web services (WS) oriented communication layer. The multi-radio networking layer generates and manages multi-radio network addresses (MRN), which are assigned uniquely to given applications in a network. This layer also provides the functionality to select the best available network based on the QoS requirements. The WS-oriented communication layer provides point-to-point communication which brings multi-radio multi-network routing to legacy SOAP messaging and group communication. This results in enhancing the SOAP API to meet the corresponding base requirements of pervasive networking environments. Generally, UbiSOAP offers a generic approach to access and use web services, while maintaining QoS and integrating heterogeneous components within the distributed environment.

4.5. (SI)$^2$

Smart items services infrastructure (SI)$^2$ (Anke et al., 2006) is a middleware developed to address the issues of platform-independent service description, deployment and invocation without any knowledge of the underlying smart items network. A smart items network may consist of RFID Systems, sensor networks, and/or embedded and mobile devices. Its architecture is mainly divided in two layers, platform-independent and platform dependent layers. Message Handlers are in the platform dependent layer and their main job is to convert the events generated by the smart items of a specific platform into platform-independent events. The Service Lifecycle Managers, also in the platform dependent layer, are responsible for deploying, starting and stopping the services on the network of smart items. The Request Processor, Service Mapper and Service Repository are the main components of the platform-independent layer. The Service Mapper is responsible for getting the service description from the middleware and searching for a feasible deployment for it. The task of the Request Processor is to handle platform-independent requests for a specified service, transmit it to the correct Message Handler and return the respective result. The general target for (SI)$^2$ is to provide an easy integration of smart items with user applications to manage and control the target environment.

4.6. RESCUE

RESCUE (Juszczyk and Dustdar, 2008) provides middleware services that allow mobile devices connected by ad hoc networks to effectively communicate among each other in critical situations such as rescue operations. RESCUE offers techniques for tackling dependency problems and for mitigating their negative effects. This is done through adding a protocol for service advertisement and discovery that reduces network traffic and propagates changes quickly. In addition, it has a subscription mechanism for real-time notifications about the availability of services. It also offers a continuously updated local database containing a global view of all available services and their metadata, while maintaining peer-to-peer communication through asynchronous service invocations. These functionalities offer the rescue teams a very flexible communications environment that does not rely on any preexisting infrastructures. As a result, RESCUE makes vital
services available to all network members in real-time and helps manage critical operations effectively.

4.7. OMG DDS with SIP

This project (Ronci and Listanti, 2010) offers and experimental implementation of a session initiation protocol (SIP) over object management group data distribution service (OMG DDS) to support interactive delay sensitive communications. Using OMG DDS, it is possible to provide QoS support, offer universal access to information and scalable efficient data distribution in addition to easy adaptation to changes. However, it does not efficiently support interactive communications such as voice and video calls over the traditional client/server model. Using SIP, the publish/subscribe service model is used to leverage the overhead and provide a better support for interactive communications. As a result, with this extension, it is possible to provide a complete efficient and scalable all IP emergency communication network. It offers an easy to use model based on topics and allows different applications to integrate and communicate using the available services over the IP network.

4.8. SOA-MM

In Groba et al. (2008) an SOA-based middleware for manufacturing systems (SOA-MM) in the form of an integration layer between the shop floor equipment and the enterprise applications is presented. Shop floor equipment may include embedded devices, RFID systems and sensors connected to the enterprise via a network. The layer addresses service granularity, physical distribution and degree of coupling individually. This is done according to the actual capabilities and data flows on the field and control layer irrespective of the type of devices used. The integration layer consists of gateway entities, core services and UDDI registry for publishing core services. A gateway entity is created to map application-specific requests to the core services for a specific device. The core services represent the main engine of this integration layer. Currently there are five core services available: device identification service, information retrieval service (applying application-specific filters to reveal relevant information from device-related information), device positioning service, structure analysis service (to reveal the structure of complex devices), and event notification service. The middleware supports business, logistics and manufacturing applications that involve varying embedded control and monitoring sub systems.

4.9. QoS SOM

An SOM to simplify the development of complex multi-organizational business applications was developed (Sheth et al., 2002). QoS SOM relies on workflow management for large multi-organizational processes. QoS management was included to achieve effective and efficient support for application QoS requirements. In addition, QoS SOM supports automatic computations of QoS based on the QoS specifications of the component web Services. In QoS SoM, the QoS requirements for the needed services are handled by the middleware to achieve best matches among available services and the application service and its quality requirements. As a result, it offers an efficient approach to service selection and integration that help achieve the business process objectives.

4.10. NINOS

NINOS (Li et al., 2010) is a distributed service orchestration engine based on light-weight agents. It operates by breaking up business processes into single activity tasks and distributing them across the infrastructure as needed. NINOS extends the PADRES (publish/subscribe applied to distributed resource scheduling) system (Yoon et al., 2011) to offer scalable distributed publish/subscribe messaging middleware to support distributed business processes. To better support business process execution, NINOS offers fine-grained content-based routing, event correlation, load balancing, fault tolerance, policy management and access to history data. In addition, it offers software brokers that help distribute processes efficiently and reduce data transfers by locating processes near the data sources. NINOS offers a procedure to map standard business process execution language (BPEL) processes, including the complete set of BPEL activities, to a set of distributed NINOS agents, with control flow realized using decoupled publish/subscribe semantics. Thus it simplifies the development of the service-oriented applications. In addition it offers better scalability than a centralized orchestration engine.

4.11. B-VIS

Burapha Vehicle-Infrastructure System (B-VIS) (Sriborrirux et al., 2009) is an SOM targeted to facilitate the utilization of distributed RFID networks for vehicle-based applications. It supports real-time vehicle fleet tracking and control and also supports road traffic monitoring systems. In addition it provides a programming model for developers to retrieve, in real-time, the data collected by the RFID networks. This reduces the complexities of the heterogeneous communication layers and data sources. The B-VIS service-oriented architecture is based on open methods and loose coupling between the software and hardware components that facilitates communication. There are two main layers, the B-Base Gateway layer and the Enterprise Application Server Layer. The middleware offers a framework that applies to various types of RFID and embedded systems where real-time data collection and processing is involved.

4.12. SAI

Service application integration (SAI) system (Parlanti et al., 2010) is a message-oriented service-oriented application development. The middleware helps in the integration of ubiquitous environments with enterprise-level services. Based on a Grid infrastructure, SAI enables information search and data mash-up in complex distributed enterprise environments with strong logical heterogeneity (i.e. having different data formats, message types, protocols, policies, etc.). It helps hide the sources and distribution of the data and resources from the user applications to achieve high transparency. In addition, the SIA features support efficient load balancing and process/service distribution. As a result, the middleware becomes capable of efficiently handling various data and information types and volumes and can scale easily as the environment grows. SIA also offers some level of system dependability using its Grid architecture that supports job failovers and load redistribution.

4.13. Weka4WS

Data mining is an area where intensive distributed computations are needed. Weka4WS (Congiusta et al., 2008) provides an SOM middleware to facilitate asynchronous data mining computations over the Grid. It is based on the Weka Toolkit designed for distributed data mining on web services resource framework (WSRF) enabled grids. Each web service in the Weka4WS framework supports operations related to WSRF-specific invocation mechanisms (createResource, subscribe and destroy) and operations requiring specific data mining tasks (classification, clustering, and
associationRules). Data mining tasks are invoked asynchronously to improve concurrency. When results are computed the client is notified. When a data mining task is started on a remote web service several steps are taken. These are: (1) Resource creation, (2) Notification subscription, (3) Task submission, (4) File transfer, (5) Data mining, (6) Results notification and (7) Resource destruction. The overall architecture enhances data mining tasks execution by allowing users to distribute the task among Grid nodes without having to wait for the results.

4.14. CROWN-C

Supporting services computing on the Grid offers a logical transition from the generally synchronous tightly coupled execution model to a highly asynchronous loosely coupled model. However, ensuring reliable integration and execution of service-oriented application on the Grid is a challenge. CROWN-C (Townend et al., 2008) is a SOM that supports the development and assessment of highly secure, dependable, service-oriented grid systems and applications based on the Globus toolkit. The main concerns here are to provide dependable services and services integration over the Grid in addition to secure the operations. In this context the SOM provides functionalities to enhance fault tolerance, support multiparty authentication tools and provide automated trust negotiation tools. One interesting fact about CROWN-C is that the middleware itself was designed and developed as a series of extensions to the original CROWN middleware instead of starting from scratch. This allowed the team to utilize existing features and functionalities like dynamic resource management, while integrating with new fault tolerance and security features.

5. SOM challenges

In general, middleware is very important for any type of distributed applications and systems. It utilizes the multi-tier architecture to create a clean separation between the applications and system logic or platforms and enhances reusability and integration of different software components. However, with the recent advances in SOC, some started to question the need for middleware since services provide some of its functionalities within the context of the service-oriented applications being developed and used. Nevertheless, we have seen through the examples we surveyed and other research in the field that middleware has proven to be of great importance to SOC. Service-oriented middleware (SOM) in particular has become an important component of SOC and several approaches and solutions were introduced. Generally, researchers and industry leaders alike accept this direction and see that it is beneficial to support service-oriented applications to combine the advantages of services with the agility and effectiveness of the middleware approach. SOM can play important roles in simplifying the development and operations of applications and systems that execute on dynamic, heterogeneous, large scale environments. Examples of these applications are telecommunication applications (Bo et al., 2010), collaborative workflow systems (Kim, this issue), emergency responses applications (Gadallah et al., 2010), business process applications (Jakkhupan et al., 2011) and distributed monitoring and control systems (Taylor et al., 2006).

The job of SOM is not only to cater for functional properties (such as communication, service creation, discovery and invocation, interoperability and abstractions) but also to support non-functional properties (such as service reliability, service scalability, service security, and QoS assurance). In Section 3 we introduced a list of the most important requirements to design a generic SOM that can support various types of environments and service-oriented application domains. However, examining these requirements closely will reveal the contradictory aspects in them. For example, security clashes directly with efficiency and scalability since introducing encryption, digital signatures, and other measures will greatly reduce the performance. Furthermore, increasing the transparency of the service integration may reduce the flexibility available for the service consumers to configure the services to their liking. In addition, considering every possible type of environment and platforms and providing the suitable abstractions to hide their details will result in a huge SOM footprint that may not be suitable to all environments and it will reduce its scalability and efficiency. There are various ways to achieve some levels of balance among these requirements and in some cases remove some of them when it is feasible for the targeted application domain.

As per the requirements, to achieve a flexible development environment it may be necessary to utilize either meta-programming languages or domain-specific languages. Such an initiative has been taken in the case of service-oriented context-aware middleware which makes use of its own developed domain-specific language to express the context-based rules (Caporuscio et al., 2007). One of the challenges facing SOM is that it is mostly based on services of heterogeneous nature and those services are distributed on heterogeneous platform. Therefore, the response time of these services is one of the factors that directly affect the application's efficiency. To improve efficiency, the response time of the services needs to be improved. One possible way to improve the response time is by introducing UDP-based SOAP transport instead of HTTP based (Lai et al., 2005). However, a typical SOM, which handles large amounts of messages and event notifications coming from numerous services and shares them among different operating components, must have a reliable system in place that ensures an uninterrupted flow of correct and required messages. This effectively contradicts with the use of some unreliable protocol for communications. Therefore, some trade-off must be considered to resolve this issue. In addition, SOM performs the job of service orchestration. Separating the application logic from the system logic by a middleware helps in such activity. An example here is the middleware based on the concept of model driven architecture (MDA) that is introduced in Sommer et al. (2009). The design patterns are techniques that help overcome common obstacles in achieving the strategic goals and benefits associated with SOC (Introducing SOA Design Patterns, 2008). However, development of middleware solutions using design patterns is still at its infancy.

As we studied the different examples of SOM, each one includes a subset of the requirements mainly based on the immediate needs the middleware is designed to satisfy. In many cases some SOM approaches may cover a subset of the requirements another approach may include; however, that does not necessarily imply the superiority of the later approach. This only shows that that particular middleware as designed for a specific target needs this particular set of requirement only. In addition, it may be the case that some approaches although covering a small subset of requirements, yet they offer more optimized utilization and operations that would enhance their performance within the targeted application domain. Table 1 provides a summary of the example SOM approaches we studied and the different features they offer and requirements they satisfy.

The main observation to make when trying to see what requirements each SOM approach supports, we found that none of them covers all of them. Yet, it may be possible to view this as a common practice since most SOM solutions are designed to solve a specific set of problems or to target a particular application domain. Therefore, it will be natural to find only those
requirements that are most needed in that domain. Another interesting observation we also came out with is that all but one (CROWN-C) of the approaches studied explicitly address the security issues or propose some security functionality as part of the middleware requirements. A few others touched on the issue, yet did not offer solid solutions or designed features to tackle security. This leaves us with an open question about the importance of security in this field. As it is well known, security is important for any type of distributed environment and SOC is no exception. SOC in particular may need stronger security measures since most services are offered over the Internet and applications using them will be exposed to a huge amount of security threats. Thus we find it interesting that it has not been adequately addressed by the SOM research community.

Understanding the requirements to meet when designing an SOM is the first step towards achieving the goal of having an efficient and comprehensive middleware solution. The idealistic version of this SOM would be able to satisfy all requirements in a very generic, yet easy to use framework. This will allow SOC application developers, providers and consumers to have all the functionalities they need while concentrating only on their business logic. However, for this to happen, there are challenges to be overcome first. Based on the different requirements we identified and how these were addressed by the different SOM approaches we studied, we could summarize these in the following:

- The overall design needs to be comprehensive and capable of offering both functional and non-functional features for the different application domains. Yet, the design should be very efficient and introduces minimum overhead on the applications using it. These two goals are contradictory and need to be tackled in a balanced way.
- The SOM should offer comprehensive, yet easy to use abstractions and APIs that will allow the application developers to find, integrate and adapt the required middleware services for their applications. This is achievable for simple distributed environments, however as the environment grows and become more divers and heterogeneous, it becomes harder to efficiently hide the details.
- The design should accommodate for future updates and changes such that developers will be able to introduce new features or modify current features to suit their needs. Exten-sibility in general is hard to achieve unless some sophisticated approaches such as modular, pattern-based, or aspect-oriented designed are used. However, these may increase the overall complexity of the design.
- The SOM should be designed and implemented in well known models that conform to current (and future) standards. This standardized approach will make it easy to adapt the solution and make it useable across a larger community. Generally, there are several models and standards used in SOC. Some of the SOM approaches we studied follow some standards, while others create their own conventions and models. As a result, even of some SOM approaches target the same goals, it may be hard to integrate services following both because of the differences in the protocols and standards used. As a result, there is a strong need for a unified approach to standardization of SOM and SOC in general to increase interoperability and generalize the utilization of available services.

6. Conclusion

In this paper we discussed service-oriented middleware, identified its general requirements and reviewed some representative approaches in the field. Then we briefly addressed some of the challenges facing the design of a generic efficient SOM. Nowadays researchers and practitioners prefer to use the SOM approach over traditional middleware because of its trait of efficiently handling heterogeneous loosely coupled SOM could support general applications, business and enterprise processes, pervasive and ubiquitous systems, mobile and embedded systems and Grid applications. For example, it is used for integrating shop floor equipment with enterprise information systems, to manage and control vehicles and traffic, to assist in disaster recovery networks, and to utilize the vast resources on the Grid for compute- and data-intensive applications. The approaches used carry some similarities in the sense that all of them agree on a common goal: to effectively compose, publish, discover and integrate services. However, the approaches used were also significantly different in terms of how to achieve these goals and what other goals they could achieve. The different SOM approaches target different features like interoperability and heterogeneity support, QoS support, adaptability, scalability, security and reliability. We discussed a number of SOM projects and highlighted their main objectives and approaches in addition to how well they satisfy the general SOM requirements. SOM is generally destined to handle the publication and discovery of web services, communication, and efficient and reliable event management among different services. In addition, SOM solutions need to support efficient handling of the heterogeneous resources and functionalities of the distributed applications. Moreover, due to the need to exchange large volumes of data, it is necessary to have a scalable, reliable and efficient system in place to ensure uninterrupted flow information and messages. It is also necessary to maintain acceptable levels of control, security and reliability through the SOM. Finally, to further enhance the approach, we must have dynamic and adaptive SOM architectures for effective integration and reuse.

References


