Int J CARS (2007)
DOI 10.1007/s11548-008-0231-8

ORIGINAL ARTICLE

Service oriented architecture for medical image processing

Mircea-Florin Vaida · Valeriu Todica · Marcel Cremene

Received: 8 January 2008 / Accepted: 26 May 2008

Abstract The aim of this paper is to present a services based architecture for medical image processing in assisted diagnosis. Service oriented architecture (SOA) improves the reusability and maintainability of distributed systems. In service oriented architectures, the most important element is the service, a resource provided to remote clients via a service contract. We propose a generic model for a service, based on a loosely coupled, message-based communication model. Our service model takes into account the possibility to integrate legacy applications. Specialized image processing services can be dynamically discovered and integrated into client applications or other services. Complex systems can be created with the help of some SOA concepts like Enterprise Service Bus (ESB) . A basic integration example is presented in the paper and the possibility to integrate dedicated scenario is specified.

Keywords Service oriented architecture · Medical image processing · Web service

Introduction

In service oriented systems, operational entities are distributed across the network in order to improve availability, performance and scalability. These entities are called *services* ~~The~~ . The service provides access to its functionality. The whole system is viewed as a set of interactions among these services. SOA promotes the reuse of services. The system evolves through the addition of new services. SOA is not tied to a specific technology. It can be implemented using a large variety of technologies, programming languages and communication protocols. Interactions between services and clients in SOA are based on a very dynamic model [1] . A service can be discovered at runtime, can be replaced if has become unavailable or can be used to create a new service (and a new functionality) . With these characteristics, SOA offers a powerful support for adaptability. The adaptability can take many forms, depending on the terminal capabilities, the network connection, etc. Microsoft has proposed a SOA based platform for healthcare [2] . Healthcare is an extremely fluid industry. Each change requires an adaptation of systems. Point-to-point integration becomes costly and complex to maintain for healthcare providers and consumers. The benefit of SOA to the healthcare industry is that it enables systems to communicate using a common framework, integration of new elements becomes less complex and the system can be adapted more rapidly.

M.-F. Vaida (✉) · V. Todica · M. Cremene
The Faculty of Electronics, Telecommunications and Information Technologies, Technical University of Cluj-Napoca, 26-28 George Baritiu Street, 400027 Cluj-Napoca, Romania
e-mail: Mircea.Vaida@com.utcluj.ro

SOA background

The term Service Oriented Architecture, SOA for short, contains some important notions. We have the following definitions for these notions [3] : /

- An **Architecture** is a formal description of a system, defining its purpose, functions, externally visible properties, and interfaces. It also includes the description of the system's internal components and their relationships, along with the principles governing its design, operation, and evolution.
- A **service** is a software component that can be accessed via a network to provide functionality to a service requester.

- The term **service-oriented architecture** refers to a style of building reliable distributed systems that deliver functionality as services, with the additional emphasis on loose coupling between interacting services.

Service

The **service** is the core element in SOA. A **service** is defined as "a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description" [4] . A service can have the following characteristics: /

- A service provides a contract defined by one or more interfaces (just like a software component) . This allows the change of the service implementation without reconstructing the client as long as the contract is not changed. Implementation details (programming languages, operating systems, ~~etc~~ etc.) of the service are not the concern of the service requestor.
- A service can be used as stand-alone piece of functionality or it may be integrated in a higher-level service (composition) . This promotes reusability. Legacy applications can be transformed in services by using some wrapper techniques.
- Services communicate with their clients by exchanging messages. Typically, the ~~request/response~~ request/response message pattern is used. From the client point of view, a synchronous or asynchronous communication mechanism can be implemented. In SOA model is not fixed a specific communication protocol. Many protocols can be used: HTTP, RMI, DCOM, CORBA, etc.
- Services can participate in a workflow (the term is *service choreography* in SOA terminology) . A workflow is "the movement of information and/or tasks through a work process" [5] and it's based on a workflow engine.
- Services need to be discovered at design time and run time by clients. This mechanism is provided by a service directory (service registry) . A service provider can publish (register) his service.

Services communicate with other services and clients using standard, dependency-reducing, decoupled message-based methods such as XML document exchanges. This characteristic is called **loose coupling** /. This term implies that the interacting software components minimize their knowledge of each other: more information is achieved at the time is needed. For instance, after discovering a service, a client can retrieve its capabilities, its policies, its location, etc. The characteristics of **loose coupling** are [6] : /

Fig. 1 SOA interaction cycle

- **Flexibility** : A service can be located on any server and relocated as necessary (with the condition to update its registry information) and clients will be able to find it.
- **Scalability** : Services can be added and removed depending on the needs.
- **Replaceability** : With the condition that the original interfaces are preserved, a new implementation of a service can be introduced, and outdated implementations can be retired, without affecting the service clients.
- **Fault tolerance** : If a server, a software component, or a network segment fails, or the service becomes unavailable for any other reason, clients can query the registry for alternate services that offer the required functionality, and continue to work in the same way.

SOA Interaction cycle

In ~~figure~~ Fig. 1 is depicted the basic case of using a service with three components: a service provider, a service requester and a service directory (service registry) . Some simple, bi-directional interactions (synchronous request/response pattern) are represented as an *interaction cycle* [7] . A real-world implementation can be more complex. A SOA architecture has three important elements:

- **Service directory** : It acts as an intermediary between providers and requesters. Usually, services are grouped by categories.
- **Service provider** : The ~~Service Provider~~ service provider defines a service description and publishes it to the service directory.
- **Service requester** : The service requester can use the search capabilities offered by the service directory to find service descriptions and their respective providers.

The service provider has to publish the service description in order to allow the requester to find it. Where it is published depends on the architecture. In the discovery the service requester retrieves a service description directly or queries the service registry for the type of service required. In this step the service requester invokes or initiates an interaction with the service at runtime using the binding details in the service description to locate, contact and invoke the service.

Enterprise service bus

The Enterprise Service Bus (ESB) is sometimes described as a *distributed infrastructure* [8] and it's a logical architectural

Fig. 2 Web services architecture

component that provides an integration infrastructure consistent with the principles of SOA. Two different issues are being addressed: the *centralization of control* , and the *distribution of infrastructure* [9] . ESB and centralize control of configuration, such as the routing of service interactions, the naming of services, and so forth. ESB might deploy in a simple centralized infrastructure, or in a more sophisticated, distributed manner. ESB does not implement a service-oriented architecture (SOA) but provides the features with which one may be implemented. ESB is not mandatory in SOA but is usually used in large (enterprise) systems with many services. The ESB might be implemented as a distributed, heterogeneous infrastructure. Minimum ESB capabilities considered in IMB view [8,9] :

- **Communications** : routing and addressing capabilities providing location transparency, administrations capabilities to control service addressing and at least one form of messaging (request/response, publish/subscribe, ~~etc~~ etc.) , support for at least one communication protocol (preferable a widely available protocols such HTTP) .
- **Integration** : support for multiple means of integration to service providers, such as Java 2 Connectors, Web services, asynchronous messaging, adaptors, and so forth.
- **Service interactions** : An open and implementation-independent service messaging that should isolate application code from the specifics of routing services and transport protocols, and allow service implementations to be substituted.

Web ~~Services~~ services (WS) standard

A web service is a special case of service, processing XML data and using communication protocols like simple object access protocol (SOAP (Simple Object Access Protocol) and HTTP [10] . Web services provide a well-defined interface that is described by an XML-based document called the *Web Service Description Language (WSDL)* document (WSDL contract) . This document contains the operations (methods) that the service supports, including data type information, and binding information for locating and communicating with the Web service operations. operations [11] . In ~~figure~~ Fig. 2 is depicted the web services architecture. The universal description, discovery and integration (UDDI (Universal Description, Discovery and Integration) plays the role of a service directory.

Fig. 3 SOAP request

Fig. 4 SOAP response

Fig. 5 Programming model

The communication protocol used with Web services is SOAP. SOAP is a protocol for exchanging XML-based messages over computer networks, normally using HTTP/HTTPS. A SOAP message is an ordinary XML document containing the following elements [12] :

- A required *Envelope* element identifying the SOAP message.
- An optional *Header* element containing generic information.
- A required *Body* element containing the ~~request/response~~ request/resp data.
- An optional *Fault* element that provides information about errors occurred while processing the request.

In ~~figure~~ Fig. 3 is represented a SOAP request message for adding two numbers.

The service response is shown in ~~figure~~ Fig. 4 . Note that the HTTP header is not represented in this example and only the required SOAP elements are used.

Model for SOA-based image processing systems

In this section we propose a model for implementing SOA-based system oriented to medical image processing. The model is generic enough to be used in other areas. The model contains a programming model, a service model and a messaging model.

Programming model

The programming model, depicted in ~~figure~~ Fig. 5 , is composed by four layers: the service layer, the component layer, the object layer and the technology layer.

Typically, a service is created using one or more components and a component is created using one or more objects. The service layer contains business services. A service is created with the help of the component oriented programming (COP) . The component layer relies on software component technologies like: Component Object Model (COM (component object model) , EJB (Enterprise Java Beans

Fig. 6 Service model

(EJB) , ~~CCM (~~ CORBA Component Model (CCM) , ~~OSGi (~~ Open Services Gateway Initiative (OSGi) or .NET ~~Component~~ Component Model. The software components can be of two types: functional components (business components) and non-functional components (like data access components, communication components or any other components) . A component is implemented using object oriented techniques (the object layer) . This layer is based on object oriented technologies (programming languages) like: C++, java, C#. Our model addresses the problem of integrating legacy applications (existent applications that are not service-based) . In order to integrate these applications, a wrapper pattern (adapter pattern) can be used. The wrapper can be applied in every layer. For instance, if the legacy application is object oriented but is not based on components, the wrapper should be applied in the component layer. If the legacy application is implemented in C, the wrapper should be applied in the object layer. This respect the proposed model: functionality is encapsulated in components, and components are created using objects.

Service model

The service model is depicted in ~~figure~~ Fig. 6 . It's composed from ~~3 layers:~~ 3 layers: the interface layer, the business layer and the resource layer.

The Service Interface Layer contains the service contract (service interface) and it's detailed in the next section. The Business Layer contains a business faç ade and business components (sometimes called functional components) . A business component performs (implements) operations described in the service contract. The business faç ade (faç ade pattern) is optional and it may be used in a complex architecture, with many business components. The resource layer contains different components (non-functional components) with the roll of interacting with external resources. In the figure are represented three of the most common types of resource access: a data access component for accessing database systems, a service gateway for accessing other services (in SOA a service can be a consumer for another service) and a wrapper (adapter) component for accessing legacy systems. The resource layer is not mandatory if the service does not use external resources. Accessing a legacy application was treated in ~~section~~ 3.1 "Programming Model" from a programming point of view. The service model is extensible, new facilities like security, transactions or QoS capabilities can be introduced.

Fig. 7 Messaging model

Messaging model

Usually, a service communicates with its clients by sending and receiving well-defined messages. A proposed messaging model is presented in ~~figure~~ Fig. 7 . A service interface is similar with an interface in object oriented programming. The service interface has the role to describe the service operations and the types of messages needed by those operations.

A message type contains one or many data types that can be translated in build-in or custom data types from a programming language. In many cases, marshalling techniques may be used to provide compatibility between server data types and client data types. Typically, this is the case when the client and the server are implemented using different technologies. For instance, an image processing service interface can describe a user defined data type (a class in object oriented programming) containing the image name, the image type, the image data (as a specific format) , etc. If the service is implemented as a web service, the data types are encapsulated (serialized) in XML documents and send over network using SOAP. Messaging exchange patterns (MEP) can be used for accessing a service. The most common access pattern used is the request/response (also known as request/reply) pattern. In this case, the service consumer sends a request to the service and receives a response. This access pattern is used in the web services applications. The client can use a synchronous or asynchronous communication mechanism. The asynchronous mechanism is preferred when communication costs are high or the network is unpredictable. Another pattern that can be used is publish/subscribe. This pattern is based on the message queue paradigm. For instance, an image capture service allows to other services or clients to subscribe to it. When a new image is captured all subscribers receives the new image. The publish/subscribe pattern is typically used with an asynchronous communication mechanism.

Experimental results

In this ~~section~~ section, we present two service implementation using web services standard and OSGi (Open services Gateway Initiative) . OSGi [13] is a java-based service platform that implements a dynamic component model (from our point of view, OSGi is a component model) .

Fig. 8 Web service component diagram

Fig. 9 Web service class diagram

Fig. 10 OSGi service component diagram

Web service example

The presented example is a basic service implementation according to our model. The service receives an image and returns a greyscale copy of that image. In a real context a more complex scenario may be integrated. In this case we consider distributed components located in dedicated library or freeware libraries and local components specific to the real application. The service interface is named *GrayscaleFilter* (~~figure~~ (Fig. 8) and has a single operation, *transformImage* ( ) . The business layer (functional layer) contains 2 components: *GrayscaleComponent* implementing the filter and *BitmapUtilsComponent* used to convert an image to byte array and vice versa.

Non-functional aspects of the service like handling incoming connections are treated by the web service used to run our service. Also, on the client side, some tools (like Visual Studio .NET) greatly simplify the work with web services by generating the necessary code to access the service. The class diagram is represented in ~~figure~~ Fig. 9 . Every component is implemented by a single class.

Note that in this simple example the business faç ade from our service model is not used and the resource layer is missing since no external resources are needed.

OSGi service example

In order to show that SOA is not based only on web services, the second example is an implementation of an image processing service using OSGi. We are using the *Knopflerfish* framework [14] as support for developing our service. In OSGi a deployment unit is called *bundle* . The framework manages the bundle life-cycle. A bundle functionality is contained typically in a jar file (java archive file) . After the bundle is created it needs to be registered in the framework and other bundle can use the published service. Our OSGi service is more complex than the web service because it needs communication facilities (offered by a communication component) because OSGi does not specify a communication protocol like a web service. The component diagram for our service is depicted in ~~figure~~ Fig. 10 .

Fig. 11 Communication component, class diagram

The service interface is called *ObjectDetectionService* and exposes a single operation, *getObjectsFromImage* . The input parameters (an image) and the return values (a collection of image objects) are not represented on this diagram. The *ObjectDetectionBundle* represent the functional part of the service (business layer) . This component uses a communication component and a gateway component. In ~~figure~~ Fig. 11 is depicted the class diagram for the communication bundle.

The component interface is called *CommunicationComponent* and provides two operations, one for sending a packet and the second for receiving a packet. A packet is a unit of information exchanged by the service. In our case the packet contains the image as a byte array. The Activator class implements *BundleActivator* / interface and is necessary in order to allow the *Knopflerfish* framework to manage the bundle (start and stop the bundle) . To be used, a bundle must be started. The bundle interface has an implementation provided by CommunicationComponentImpl. The communication is based on standard sockets (with the help of ServerSocketListener and SocketHandler) . For this service, the resource layer contains a component (*ServiceGatewayBundle*) for accessing other services. The object detection algorithm implemented needs to use a grayscale image in order to provide good results. This component contains the logic to access our grayscale web service presented in ~~section 4.1.~~ "Web service example".

Conclusions

In this paper, we have proposed a model for implementing SOA-based image processing systems. The model contains a programming model, a service model and a messaging model. We have focused on the concept of service. The service is represented as a layered architecture with a service interface layer, a business layer and an optional resource layer. The service interface layer contains the service contract (service interface) . The business layer contains the service functionality, contained in business components. The resource layer contains non-functional components, used to access external resources like database systems, other services or legacy applications. Service Oriented Systems are very flexible. A service can be discovered at runtime, can be replaced if is unavailable or can be incorporated in a new service (a powerful support for adaptability) . Our future goals are to create a SOA based platform for adaptation with applicability in medical domains. This platform may be based on ESB in order to provide full SOA facilities. The experience of different teams in medical image processing and distributed applications will be ~~used,~~ used [15,16] .

Acknowledgments The research activity will be supported by PN2 IDEI project, no. 1083 and no. 1062 from National Authority for Scientific Research that will be realised till 2010.

References

1. Herrmann Muhammad Ahtisham Aslam, Oliver Dalferth, M, Aslam MA, Dalferth O (2005) Applying Semantics (WSDL, WSDL-S, OWL) in Service Oriented Architectures (SOA) /. Universitä t Leipzig, Germany, Technical report
2. Microsoft Healthcare (2007) [online] /. http://www.microsoft.com/industry/healthcare , (2007)
3. Treadwell J (2005) Open Grid Services Architecture Glossary of Terms. grid services architecture glossary of terms. Hewlett-Packard
4. Organization for the Advancement of Structured Information Standards (OASIS) , Service Oriented Architecture (SOA) (2006) Reference Model, Public Review Draft 1.0
5. Workflow definition (2007) [online] /. http://en.wikipedia.org/wiki/Workflow (2007)
6. Srinivasan L, Treadwell J (2005) An Overview of Service-oriented Architecture, Web Services and Grid Computing, overview of service-oriented architecture, web services and grid computing. HP Software Global Business Unit
7. Haller A, Gomez JM, Bussler C (2005) Exposing Semantic Web Service Principles semantic web service principles in SOA to Solve solve EAI scenarios
8. Robinson R (2004) Understand Enterprise Service Bus enterprise service bus scenarios and solutions in service-oriented architecture, IBM
9. Patterns: Implementing an SOA Using an Enterprise Service Bus (IBM Redbooks) , IBM.Com/Redbooks, (2004) IBM.Com/Redbooks
10. Specifications WS (2007) [online] /. http://www.w3schools.com/webservices/default.asp , (2007)
11. Jeffrey Hasan Expert Service-Oriented Architecture in C#: Using the Web Services Enhancements 2.0, Apress, 2.0. (2004) Apress
12. Specifications SOAP (2007) [online] /. http://www.w3schools.com/soap , (2007)
13. OSGi Service Platform The Open Service Gateway Initiative, Initiative (2003) IOS Press
14. OSGi Tutorial A Step by Step Introduction to OSGi Programming Based on the Open Source Knopflerfish OSGi Framework Sven Haiges, Haiges (2004) http://www.knopflerfish.org/tutorials , (2004)
15. Mircea-Florin VAIDA, VAIDA (2007) Valeriu TODICA Dynamic Adaptability of Image Processing Components in Medical Applications, ISSCS 2007, dynamic adaptability of image processing components in medical applications, ISSCS. Iasi 2007, Vol. vol 2, pp. pp 381–384
16. Cremene M, Riveill M (2007) Service-context knowledge-based solution for autonomic adaptation. In: Proceedings of the 4th International Conference international conference on Autonomic and Trusted Computing autonomic and trusted computing (ATC-07) , Hong Kong, China, China. Springer Verlag, Hong Kong, pp 11–13