Reconciling Component Based & Service Oriented Software Engineering: Application to e-Health System

Bentlemsan Khadidja
The National school of computer
Sciences (ESI)
Oued smar, Algiers, Algeria
k_bentlemsan@esi.dz

Bennouar Djamel
Computer Science Department
Akli Mouhand OulHadj University
Bouira, Algeria
dbennouar@gmail.com

Hidouci Walid
The National school of computer
Sciences (ESI)
Oued smar, Algiers, Algeria
w_hidouci@esi.dz

***Abstract*— today, with the rise of the internet technology, software systems need to be dynamic, highly flexible and at the same time controllable and simple to maintain. To achieve this goal, recent studies have mixed the strength of Component Based Software Engineering (CBSE) and Service Oriented Software Engineering (SOSE). In the present paper, we show the importance of such collaboration through a critical e-Health case study: The Organ Transplant Management System (OTMS).**

Keywords- Component Based Software Engineering, Service Oriented Software Engineering, Collaboration, e-Health.

INTRODUCTION

Component Based Software Engineering was a shift of paradigm from traditional software development to simplify design of software, improve quality and reduce costs of development by ensuring the reuse of pre-existing software packages known as components. Component is a part of system, available in ready for use state that can communicate with other components through interfaces according to well define Architecture Description Languages. Despite the success of CBSE, it does not address all complexities that software developers are facing today especially with the emergence of internet; heterogeneity of platforms and protocols, and the difficulty of locating and selecting components against system requirements, have led to the emergence of new paradigm know as Service Oriented Software Engineering .

SOSE utilises services as functionality units, a service is a black box entity done by a provider to complete desired end results for a consumer. SOSE ensure loose coupling in order to minimize the dependencies and thus to reduce the risk that a change in one part of an application will force changes in other parts[9].SOSE simplify the integration of distributed systems build on various operating systems and technologies but still face challenges as automatic composition.

Although they are built around similar concepts and share many characteristics, each paradigm has its own philosophy, abstraction, issues and challenges. It would be beneficial to combine the force of the two paradigms to simplify the architecture and design of large-scale and distributed systems.

In other side, e-Health systems are facing the challenge for improving quality, efficiency and safety of patient's data especially with systems that have "life and death implication". For meeting this challenge, we propose in this paper to mix CBSE/SOSE concepts and applied them in Organ Transplant Management System.

The remainder of the paper is organized as follows; (Section 2) presents the main characteristics of CBSE and SOSE and a brief comparison between them. (Section 3) outline related works in the area; (Section 4) shows the Organ Transplant Management System by combining the strength of CBSE & SOSE, the last section (Section 5) concludes this paper.

CHARACTERISTICS OF CBSE AND SOSE

Component based software engineering is being proposed for building high quality, evolvable, large-scale systems in a timely and affordable manner through assembling existing components together with well-defined software architecture. Although there are many different views on what a component is, the common consensus regards a component as (1) An independent part of a system that fulfils a precise function, (2) Has a specific behaviour and communicates with other components through its interfaces that describe service required and service provided, and composed of a number of interaction points called ports or players, (3) May be reused in different contexts and without knowledge of its internal

structure. (4) Works in the context of a well defined architecture called component model. Currently, various component models exist; the majority of them are targeted toward specific application domains, they are coming either from the industry such as JavaBeans [16] to build user interfaces, CCM [17] for the construction of application servers, or from research teams as Fractal [15] that aims to be more generic model. In other ways, Service-oriented software engineering provides methods for building software systems by supporting loose coupling. In SOSE functionality is seen as a collection of interoperable services. A service is a self-contained function with a well-defined interface that does not depend on the context or state of other services. The communication can involve either simple data passing or it could involve two or more services coordinating some activity. SOSE describes the interaction among three actors: service provider, service requester and service registry. The service provider defines services and publishes their interface descriptions in the service registry in order to make them discoverable. The service registry contains service descriptions and references to service providers. The service requester is a client, either an end-user application or other service. It searches in the service registry for a specific service using the service description and calls the service provider when a matching service is found. The Web services and the OSGi [10] platform are the most likely technology of SOSE. Web services essentially use XML to create a robust connection through WSDL, SOAP and UDDI; WSDL (Web Services Description Language): is an XML-based language for describing Web services and how to access them. SOAP (Simple Object Access Protocol): is a simple XML-based protocol to let applications exchange information over HTTP. UDDI (Universal Description, Discovery and Integration): it is a directory for storing information about web services described by WSDL. In other ways, a service in OSGi is published as a service interface, an object implementing the service and a set of properties. The properties, defined keys and values and allow differentiating services that are registered under the same interfaces. In addition, the service registry supports a notification mechanism that allows bundles to be notified when a service is registered or unregistered.

CBSE and SOSE are similar to each other in many points. Both are interface based, they share the same [13] objectives and principals as modularity, reusability, rapid development of software system and maintainability. But still have distinctive characteristics and mechanisms grouped in the following points:

- **Encapsulation type:** CBSE support a variety of encapsulation types ; black box exposing just the interface , grey box exposes interface and a part of implementation with the possibility of making modifications in the exposed part, and white box exposing interface with the full implementation that can be changed as needed , in contrast of CBSE, SOSE support just black box.
- **Interoperability**: it is the ability to build software systems by assembling pieces of different nature: In CBSE, components must have the same nature and follow the same component model. , SOSE allows the composition of services of different nature turned in different platforms.
- **Behavioural model**: SOSE deal with service behavioural model either at lower level of abstraction, tightly coupled with the underlying programming languages, or behaviour is described at higher levels of abstraction than programming languages. In component based approaches authors describe behaviour at architectural level [18].
- **Binding:** components in CBSE are connected at the design phase in contrast of SOSE where integration depends on external runtime factors (dynamic composition in SOSE still be a complex task).
- **Dynamic availability:** SOSE support the dynamic availability which means that service can appear and disappear at execution time, that characteristic is not support by CBSE.

Consequently, building enterprise applications using purely component based or service oriented software engineering is not sufficient. It would be necessary to mix the strength of both paradigms to address issues that each paradigm can't solve alone.

RELATED WORK

Recent studies have coupled the strength of CBSE and SOSE ,although they focus on different objectives , principals are the same, we present briefly studies related to our work.

.

- Beanome: The authors [1] have found that OSGi specification is limited to the definition of a service gateway and does not cover a sophisticated component model to build complex applications. They add a lightweight layer on top of the OSGi framework that implements a simple component model. Beanome, however,

does not provide support for dynamic changes.

- Gravity : [2] defines a component model where component provide and required service. Gravity introduces a new approach known as service oriented component model, In Gravity, an execution environment entity, called the Service Binder adapts component instances and compositions with respect of dynamic changes.
- FROGi (Fractal components deployment over OSGi): is an extension of Fractal component model which has been proposed for two reasons; the first one is to offer a flexible component model to the OSGi's developers to simplify bundles development. The second reason is to leverage the OSGi's deployment capabilities to package and deploy Fractal components. FROGi [4] is implemented on the top of the OSGi Platform by combining with Julia, the Java-based reference implementation of Fractal component model.
- SOFA2: this work exposes a solution that provides dynamic availability and discovery by introducing SOFA2 [15] into OSGi platform. It proposes to use a proxy that handles method invocations and acts as a mediator between component interfaces and matching services. The services behind the proxy may appear and disappear dynamically. It uses also aspect-oriented controllers and annotations. The annotations serve for specifying service-enabled components and interfaces in a declarative way, while the aspects provide components with the desired functionality through the corresponding OSGi controllers. In this way, SOFA 2 components can both access and publish OSGi services.
- injected POJO (iPOJO): iPOJO [5] is an extensible component model based on the POJO principles and implemented also on top of the OSGi service platform. One of the main goals of iPOJO is to keep service-oriented component development as simple as possible. The code of a component should focus on business logic, not on non-functional requirements. To achieve this objective, iPOJO provides a component container (handlers) that manages all service-oriented component aspects, such as service publication, service object creation, and required service discovery and selection. Moreover, iPOJO containers are extensible to support other non-functional requirements as configuration, persistence, security.
- SCA (Service Component Architecture):[9] is a service-oriented component model where an application is constructed of components that can be hierarchically composed of other components. Each component typically implements some business logic, exposed as one or more services. A service, provides some number of operations that can be accessed by the component's client. SCA itself is technology-neutral and aims to support a wide range of implementation technologies (BPEL, Java, and C++). SCA bindings are generated at runtime during the component composition. Moreover, there is a capability to create bindings dynamically by components themselves. Actually, there are several implementations of the specification such as Apache Tuscany SCA [9], IBM WebSphere [8] or FraSCAti [7].

ORGAN TRANSPLANT MANAGEMENT SYSTEM

In the last years, organ transplantation has played an important role in the treatment of patients with end-stage diseases of most major organ systems. Treatment of patients through the transplantation of organs is one of the most complex medical processes currently carried out. This complexity arises not only from the difficulty of the surgery itself but also from extra issues as lifetime of organs (such as heart, lung, intestine, liver, pancreas, kidney) deteriorate rapidly between when they become available and implantation (becoming useless in less than 6 hours in some cases).

The scenario of OTMS is the following [3]: when a donor becomes available, he is assessed for potential donations by the duty transplant surgeon. Data from medical records and examinations are passed to the medical laboratory to carry out potential matching tests to the Organ Transplant Authority to begin the search for a match. After making a check for extremely urgent cases, the transplant manager begins a round robin process following established matching criteria. Hospital with potential recipients are contacted in order decide whether or not an organ could be assigned to one of their patients. Medical analysis results are used to inform this process where available. Once a decision is made, the recipient is contacted and prepared. In most cases a team from the Hospital at which the recipient will have the organ implanted will travel to the Hospital where the extraction will take place to perform the extraction and subsequently return with the organ to the

implantation site. The implantation surgery is followed by post care.

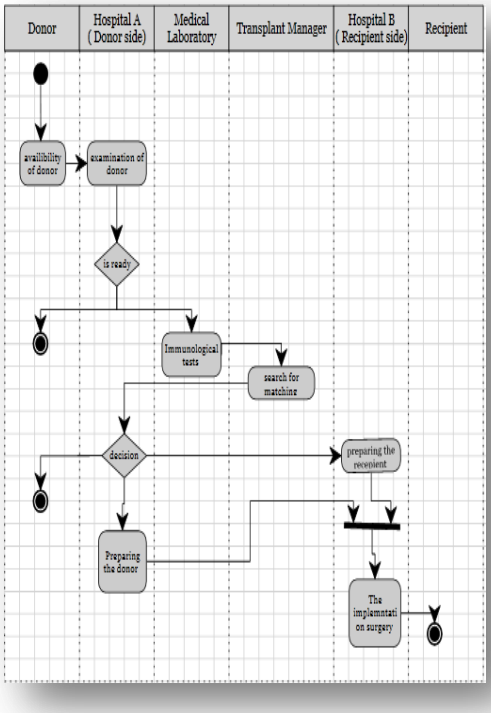

Figure1: UML Activity Diagram of OTMS

e-Health systems like Organ Transplant Management are difficult to develop due to their complex and decentralized nature. Also, Interoperability in such system is one of the major concerns. It is difficult to design exact and flexible interoperable architecture which transmit data and exchange information between systems to systems (hospital to hospital).

The Service Oriented Software Engineering facilitates the development of such systems by supporting modular design, application integration and software reuse that helps to exchange the information between similar and dissimilar applications (interoperability) .

OTMS can be seen in this approach as a complex web services (see Figure 2) composed of four basic web services (Medical analysis Controller, Recipient Selector, Patient Matching and Surgery Order) , Table 1 shows the interface and the role of each basic web service;

Web service	Function
File Medical Analysis Controller (**Patient** p)	Gives the result file of the corresponding Patient p from Medical laboratory
List< Patient > Recipient Selector (**List< Patient >** list**)**	Selects the list of extremely urgent cases
Boolean Patient Matching (**Patient** donor, **Patient** recipient)	Releases the match between the donor and recipient according to immunological tests results
Boolean Surgery Order(**Patient** donor, **Patient** recipient)	When the donor and the recipient are ready for the surgery , this web service orders the operation

Table 1: basic web services in OTMS

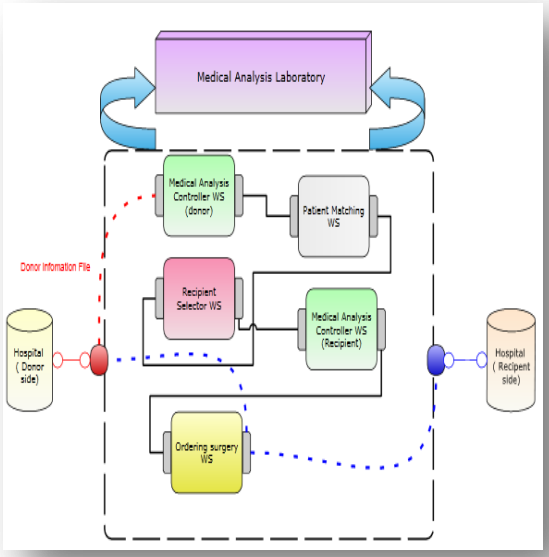

Figure2: OTMS as a Web service

Disscusion : From a security perspective, data accuracy and privacy are key issues in e-Health system and without these; no solution will get any real success.in other hand , web services with their interoperable nature present a favorable points to attacks. An obvious security requirement is the need to control access to OTMS. many practical techniques already exist. Thus , OMTS is considerated as a highly insecure system. In which time is a matter of life or death, we must adress its issues with **rapid** solutions.

For this reason , we must take full advantage of SOSE concepts as modularity and maintainability, and as CBSE shares these concepts with SOSE, we propose to consider security aspect as a separate component deployed according to CBSE concepts , we can use any component model to implement this idea (for example Fractal).

Security component can be updated at any time, we can replace a security Algorithm by another without touching the initial artchitecture.

Security component can be composed of a set of primitifs components.

This component identifies the sensitivity of information within a specific data item and then restrict access to a user base in accordance to their predefined roles or identities.

Extending this idea directly to our OTMS, we obtain the following architecture :

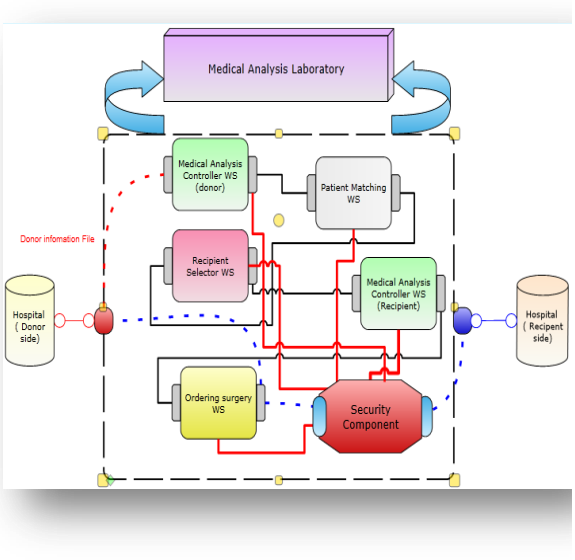

Figure3: OTMS as heterogeneous component

By this manner, OTMS is a complex component composed of a set of hetergenious components (security component and web services) simple to maintain and the modularity is not violated.

Thus, web services and components are treated at same level , this idea can be explored to define new hybrid approches.

CONCLUSION AND FUTURE WORKS

In this paper we have discussed the importance of the collaboration between Component Based and Service Oriented Software Engineering to develop dynamic, highly flexible and controllable applications and have cited some related works in this area.
Nowadays there is a lack of systems using web services and Component Based software engineering concepts at the same time, Most works are specific to OSGi, they add a component model on the top of OSGi Platform as a layer, the service and component are not manipulated at the same level of abstraction. So the construction of hetergenious application composing at the same time CBSE and SOSE are not possible.
Our intention in this paper is to show how it can be practical to apply such fusion (web srevice and compoents) in critical e-Health cases as Organ Transplant Management System.
Presently, we are working on the definition of a hybrid approach between CBSE and SOSE called HACS that consists of Component Model, Connector Model and Architecture Description Language.
In future we would like generalize this idea to other e-Government systems like e-Learning and deploy them into HACS approach.

REFERENCE

[1] H. Cervantes, and R. S. Hall., “Beanome, A Component Model for the OSGi Framework,” Proceedings of the workshop Software Infrastructures for Component Based Applications on Consumer Devices, Lausanne, 2002

[2] H. Cervantes, and R. S. Hall, “Autonomous Adaptation to Dynamic Availability Using a Service-Oriented Component Model”, Proceeding of the workshop Software

Infrastructures for Component Based Applications on Consumer Devices, Lausanne, 2002.

[3] Álvarez, S., Vázquez-Salceda, J., Kifor, T., Varga, L. Z., & Willmott, S. (2006). Applying provenance in distributed organ transplant management. In Provenance and Annotation of Data (pp. 28-36). Springer Berlin Heidelberg.

[4] M. Desertot, H. Cervantes and D. Donsez, "FROGi: Fractal components deployment over OSGi", Software Composition, volume 4089 of Lecture Notes in Computer Science, pages 275–290. Springer 2006.

[5] C. Escoffier, RS. Hall and P. Lalanda, "iPOJO: an Extensible Service-Oriented Component Framework", 2007 IEEE International Conference on Services Computing (SCC 2007).

[6] J. Estublier,. Vega, G." Reconciling Components and Services: The Apam Component-Service Platform", 2012 IEEE Ninth International Conference On Service Computing.

[7] Apache Software Foundation. The Apache Tuscany Project, 2008 http://tuscany.apache.org/.

[8] IBM Webspher http://www01.ibm.com/software/webspher e/

[9] L. Seinturier, P. Merle, D. Fournier, N. Dolet, V. Schiavoni, and J-B. Stefani. "Reconfigurable SCA Applications with the FraSCati platform". In SCC '09: Proceedings of the 2009 IEEE International Conference on Services Computing, pages 268–275, Washington, DC, USA, 2009.

[10] OSGi Alliance: OSGi Technical Whitepaper. (March 2012) Release 5) http://www.osgi.org.

[11] M. Jiang, and A. Willy, "Architecting Systems with Components and Services", Information Reuse and Integration, 2005.

[12] Apache Felix, http://felix.apache.org/

[13] H. P. Breivold and M. Larsson, "Component-Based and Service-Oriented Software Engineering: Key Concepts and Principles," In Proc. of SEAA 2007, Lubeck, Germany, Aug 2007.

[14] Fractal, http://fractal.ow2.org/

[15] SOFA2, http://sofa2.ow2.org/

[16] S.Bodoff, E.Armstrong, J.Ball, D.CarsonThe J2EE Tutorial. Addison-Wesley (2004) 2nd edition.

[17] J.Siegel: CORBA 3 Fundamentals and Programming. 2nd edition. Wiley (2000).

[18] A. Causevic, A. Vulgarakis, "Towards a Unified Behavioral Model for Component-Based and Service-Oriented Systems", Computer Software and Applications Conference, 2009 (COMPSAC '09).