Service-oriented Software Development and Management: A CASE Tool-based Approach

Youcef Baghdadi
Sultan Qaboos University/Department of Computer Science, Muscat, Oman
Email: ybaghdadi@squ.edu.om

Bashar Alani and Zuhoor Al-Khanjari
Sultan Qaboos University/Department of Computer Science, Muscat, Oman
Email: bashar.alani@gmail.com, zuhoor@squ.edu.om

***Abstract*—In order to realize Service-Oriented Architecture (SOA) with Web Services (WSs), a new engineering approach is required; we refer to it as Service-Oriented Software Engineering (SOSE). This approach would sketch out a method, including a process, models, languages, tools, and notations. It encompasses three dimensions: (i) services, (ii) compositions, and (iii) management of both services and compositions. The existing methods have not considered the three perspectives neither they have considered how these perspectives can be achieved together through a comprehensive approach. This paper presents an architecture, a specification, and an implementation of a CASE tool for developing and managing Service-Oriented Software (SOS). The CASE implements: (1) a metadata that represents WSs from four perspectives: description, deployment platforms, legacy systems, and composite software, and (2) a set of management artifacts built on top of the metadata. The metadata is implemented as repository, the core component of the CASE tool architecture.**

***Index Terms*—Web Services; Service-Oriented Software; Approaches; Metadata; CASE Tool**

I. INTRODUCTION

SOA is an architectural style that promotes flexibility and agility through abstraction, separation of concerns, loose coupling and interoperability of its components. WSs technology constitutes a suitable distributed computing platform [1] to realize SOA [2].

However, realizing SOA with WSs as reference architecture for SOS requires a new engineering approach; we refer to as SOSE. This approach should not only deal with services as basic components, and software solutions as composites, but more importantly with the management of both services and composites. Indeed, the management is a complex task and a serious challenge, due to the growing number and types of WSs deployed, and the changing and flexible nature of the composites built on top of them. Such a complexity requires tools that automate and support the management and monitoring efforts [3], [4], [5], [6], [7], [8], [9], [10], 11].

Several methods from both academia and industry have been developed [12], [13], but neither they have considered comprehensively the three perspectives, nor have they showed how these perspectives can be achieved through a unique comprehensive approach.

Therefore, we advocate for a new CASE tool-based approach geared towards rapid generation and management of composites from existing services. The existing CASE tools such as IBM Rational Rose [14] generally deal with generating components from a specification by wrapping existing code into WSs.

The proposed CASE tool, by embedding existing generators and applications wrappers, can embrace any of the well-known approaches: top-down, bottom-up, or their extensions: green-field and meet-in-the-middle specifically devised to cope with WSs. These approaches play on the service contract and logic, i.e., whether a contract already exists (there exists a specification of the service) or is newly designed, and (ii) whether the logic already exists (there exists a piece of code implementing the service) or to be developed [15], [16].

This paper presents an architecture, a specification, and an implementation of a CASE tool for WSs development, deployment, composition, and management. It is based on: (1) a repository implementing a metadata that represents WSs from four perspectives, namely WSs description, WSs deployment platforms, wrapping legacy systems into WSs, and composites, and (2) a set of management artifacts built on top of the metadata. These artifacts are required to: (i) evaluate WSs quality individually [9], (ii) provide rules governing the control of Business Process Execution Language (BPEL) [5], and continuous inspection of the software built out of the services [2], and (iii) align service-based solutions with business goals [6].

The remainder of this paper is as follows: Section 2 details the metadata of the WSs. Section 3 presents the management perspective. Section 4 presents the architecture of the proposed CASE tool. Section 5 details its specification. The implementation is provided in Section 6. Section 7 presents related work. Finally, a conclusion section presents further developments.

doi:10.4304/jetwi.4.4.371-378

II. WSs METADATA

WSs metadata, implemented as repository, is the core component of the CASE tool. The metadata emphasizes four perspectives of the WSs as shown in Figure 1. These are:

(P1) Description of WSs with WSDL2

(P2) Wrapping of legacy systems and applications

(P3) Deployment of WSs within their servers

(P4) Development of composites

The objective of having these perspectives is twofold:

1. Guidance towards the realization of SOSE
2. Management and monitoring of the components and composites as main IT assets

A. WSs Description Perspective

The description of a WS should be specified with a machine-readable language. It concerns with: (i) the functional and non-functional requirements of the service, i.e., what functionality a service provides, (ii) the communication style, i.e., how it communicates, and (iii) its locations on the Web, i.e., where to find it. Accordingly, a description language such as WSDL1 or WSDL2 expresses three distinct parts:

1) The abstract part expresses the interface of the service.

2) The concrete binding expresses the communication aspect and style. The concrete part references (imports or includes) the abstract part.

3) The implementation part expresses the location of the service. The implementation part references (includes or imports) the associated concrete binding.

These three parts are related to each other by inclusion relationships as distinguished in the metadata as shown in Figure 1.

B. Services Deployment and Registration Perspective

The metadata, represented with a UML class diagram in Figure 1, distinguishes the services deployment perspective. Services are deployed within servers (i.e., run-time servers). A server can run as standalone, as Web server, or within an application run-time server provided by an environment such as J2EE or .NET. A service server consists of a service container that manages the life cycle of the application implementing the services, and a SOAP processor that processes the exchanged messages.

The service container is responsible for the following:

- Management of the lifecycle of the application that implements the service
- Generation of the WSDL document, that will be registered within the Universal Description and Discovery Interface (UDDI), in which the client applications can find it and generate a client proxy. At run time, the client uses the proxy to construct and send SOAP message to the services

The SOAP processor is responsible for the following:

- Processing of the incoming messages
- Conversion from XML into native Programming Language (PL) data types
- Routing of the request to the application that implements the service

C. Legacy Systems Wrapping Perspective

In SOSE, designing and developing a service consists mostly in wrapping candidate functionalities. Generally, these functionalities exist in the legacy systems, however if some functionalities do not exist, we need to develop them as services. These services are then registered in a private/public registry. Most of the applications do exist in the current system. They will be considered as legacy software applications. These applications are still in use and need to communicate in a distributed heterogeneous environment. The UML class diagram in Figure 1 distinguishes the service wrapping perspective.

D. Composition Perspective

Reusability is one of the major properties of services since services can be reused instead of being redeveloped. Therefore, WSs stack is seen from a usage point of view to provide a flexible software composition implementing Business Processes (BPs). The UML class diagram in Figure 1 distinguishes the services composition perspective, where a composite is presented and modeled, with respect to SOA, by using BPEL (the partners are WSs).

III. MANAGEMENT PERSPECTIVE

Due to the growing number and types of WSs deployed as well as the changing and flexible nature of the composites, the management and monitoring of both service and composites becomes a complex task and a serious challenge. Such a complexity requires tools that automate and support the management and monitoring efforts. Indeed:

- From a service perspective, the ability to evaluate WS quality individually is critical towards the achievement of the service oriented computing paradigm [9].
- From a composition perspective, not only we need rules governing the control of BPEL [5], but also a continuous inspection of the solutions built out of the services [2].
- From a business perspective, the management would allow alignment of service-based solutions with business goals, which requires the assessment of the impact of service execution [6].

Although, process-monitoring capabilities are provided by toolsets in platforms for developing, deploying, and managing service applications (e.g., Oracle WebLogic, Vitria BusinessWare), we still need these tools to be embedded within a larger CASE tool that provides the necessary metrics and statistics for both

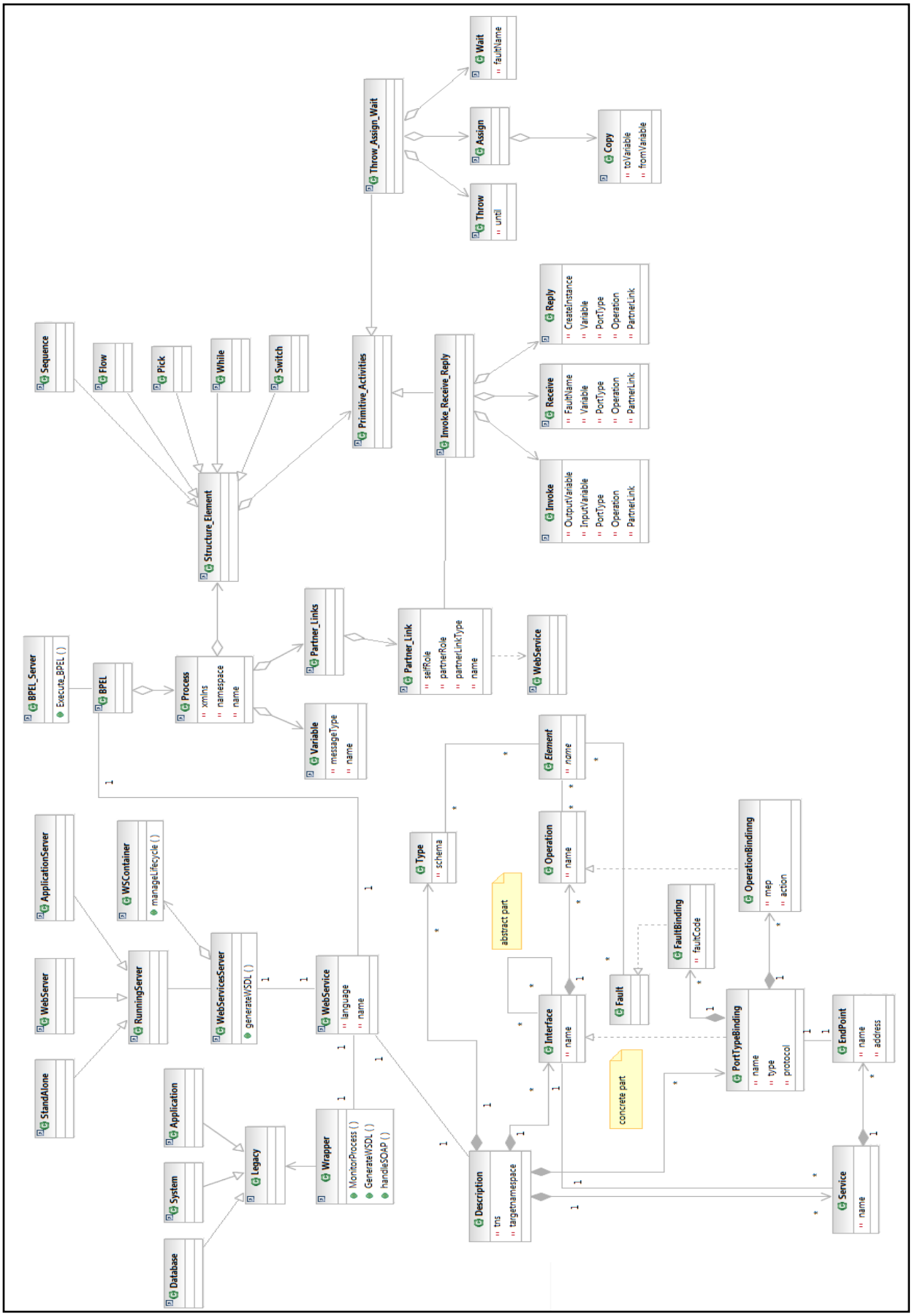

Figure 1. UML class diagram for WSs metadata

basic services and compositions. These metrics and statistics concern with quality and performance parameters, namely the workloads and the configuration of: (i) the WSs servers, (ii) the WSs containers, (iii) the SOAP engine, and (iv) the composites supporting BPs. The performance of the WSs as components and the composites supporting the BPs are depending not only on the performance of the WSs themselves, but also on the underlying platform.

IV. CASE TOOL ARCHITECTURE

A. Definition and Role

CASE tools assist system developers to meet the challenges they face in their work. These tools provide an automated environment to design and implement system projects. The new generation of CASE tools enables system developers to improve both quality and efficiency of their system, resulting in a net improvement in maintenance and development productivity [17].

The proposed CASE tool presents different perspectives of a WSs stack to come up with a unified vision of management. It has two main functionalities: (F1) guidance towards SOS development, and (F2) control of the quality and performance of WSs and the composites

F1. Guidance towards SOS development: The CASE tool assists developers to:

- Develop WSs by using any of the well-known approaches such as top-down, green-field, bottom-up, or meet-in-middle
- Develop BPEL specification of executable and abstract BPs

F2. Control of the performance: The CASE tool is used to have control over:

- Performance of WSs, the main building blocks of SOSE, with respect to an efficient usage in composites
- Performance of the platform, i.e., the involved servers, namely WSs server, WSs container, and the SOAP engine with respect to their workloads
- Auditing, monitoring, and troubleshooting
- Dynamic WSs provisioning such as:
  - Provisioning WSs to authorized requesters
  - Dynamic allocation/de-allocation of severs
- WSs lifecycle/state management:
  - Exposing the current state of a WS
  - Managing life cycle, including the ability to start and stop a WS, the ability to make configuration changes to deployed WSs
  - Supporting the versions of WSs
- Performance of composite software

B. CASE Tool Architecture

The CASE tool architecture is made up of the following components that are represented as UML packages as shown in Figure 2:

(C1) Repository component that represents the metadata. It has other packages as shown in Figure 3

(C2) Performance component

(C3) Management and monitoring component

(C4) SOS development

(C5) Utilities component

These components are related to each other as follows:

(R1) The performance component depends on the repository.

(R2) The management and monitoring component uses both repository and performance components.

(R3) The SOS development uses both repository and utilities components.

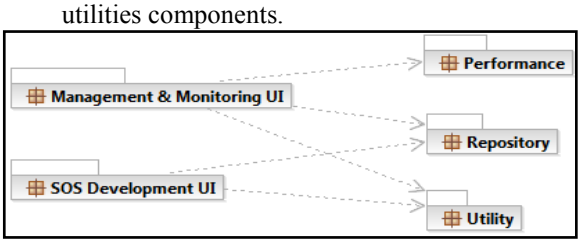

Figure 2. Architecture of the CASE tool

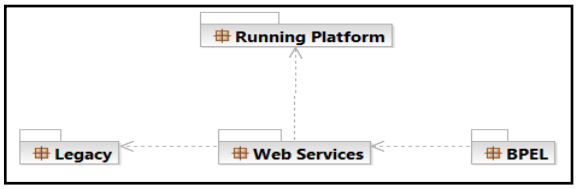

Figure 3. WSs repository component

Repository component

The WSs repository package expresses the repository related to the running WSs, the platform, the legacy, and the composites. Therefore, it has four packages as shown in Figure 3. The BPEL package depends on the WSs description package, which, in its turn, depends on the platform package.

Performance component

This component expresses the performance parameters, the workloads and the configuration of: (i) the WSs servers, (ii) the WSs containers, (iii) the SOAP engine, and (iv) the composite software. It is worth noting that the four subsystems are depending on each other because the WSs containers and the SOAP engine are depending on the WSs server, and the WSs servers depend, in their turn, on the Web/Application servers though they may be standalone servers. The performances of the WSs as components and the composites as software are depending not only on the

performance of the WSs themselves, but also on the underlying platform.

Management and monitoring component

The management and monitoring expresses the management use cases and the collaborations realizing them. It contains four other interrelated packages:

P1. WSs server package dedicated to the management of the WSs server

P2. WSs container package dedicated to the WSs container

P3. SOAP engine package dedicated to the management of the SOAP engine

P4. Composite dedicated to the performance of the BPEL describing the composites

The WSs containers and the SOAP engine are depending on the WSs servers, which, in their turn, depend on the Web/Application servers.

SOS development

SOS development assists the developer of WSs and composites. The composites are software represented with BPEL, which itself uses WSs are partners. It mainly allows:

- Developing, deploying, and maintaining WSs by using any of the well-known approaches: top-down, bottom-up, green-field, or meet-in-the-middle. It is worth noting that the CASE tool will embed other tools such as WSDL generators and wrappers.
- Developing, deploying, executing, and maintaining BPEL that specifies composite software. The CASE tool will embed a BPEL engine for this purpose.

Utilities component

The utilities component is made up of a set of auxiliary tools such as WSDL generators, wrapper tools, and engines (e.g., BPEL engine), and statistics tools.

V. CASE TOOL SPECIFICATION

The CASE tool is specified with:

- A set of Use Cases that provide value to different types of actors
- The collaboration and sequence diagrams realizing the use cases

A. The Actors

Different types of users may use the CASE tool as shown in Figure 4. A user may be an administrator or a developer. The developer may be a WS developer or a BPEL developer.

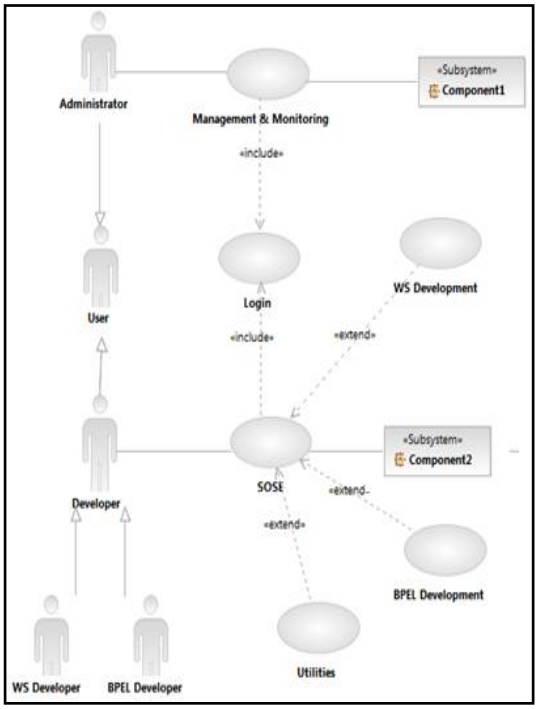

Figure 4. Different types of actors

B. The Use Cases

The CASE tool realizes different use cases for the different types of actors. There are two types of use cases:

1. SOS development use cases
2. Management and monitoring use cases

SOS development Use Cases

The CASE tool assists the developers of SOS to develop the WSs and composites. For this purpose, the CASE tool will embed wrappers, generators, and BPEL engines. The tool enables:

1. WS development by using any of the aforementioned development approaches. Figure 5 shows the different use cases from the perspective of WS development. It mainly consists in:
   - Develop contract
   - Generate WSDL document
   - Wrap legacy
2. The composite development consists mainly in developing BPEL as shown in Figure 6. The main use cases are:
   - Generate BP
   - Create BPEL
   - Execute BPEL

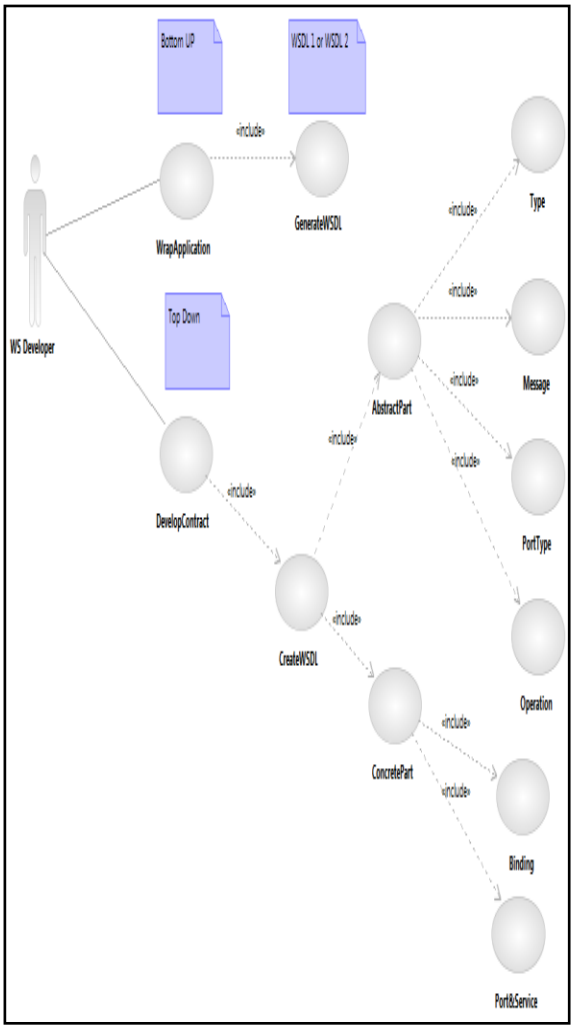

Figure 5. WSs development use cases

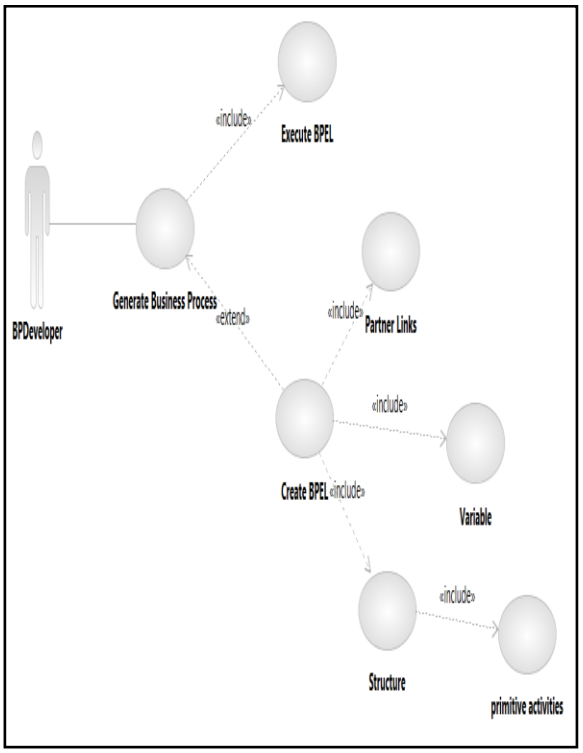

Figure 6. Composite development use cases

Management and monitoring Use Cases

The CASE tool is used to manage and monitor: (i) the deployed WSs, (ii) the BPEL describing the composites, (iii) the Web/Application servers, (iv) the WSs servers, (v) the WS containers, and (vi) the SOAP engine as shown in Figure 7. This makes the WSs stack flexible and configurable, which allows any change in the WSs, their platform workloads and configuration, and the BPEL describing the composite. It concerns with:

a. The performance and the quality of the deployed WSs as running applications: the required parameters may be configuration, workload, quality assurance, or statistics. The quality assurance parameters include the quality of services, security, dependability, cost, billing, and maintenance. The statistics are related to performance such resources consuming (e.g., CPU time, communication time, memory); and the use of the WSs such as the number of clients using the service by unit of time, the number of running copies of the service and so on. The BPEL management concerns with information such as: type of BPs (internal, crossing), the workflow, and the composition out of the WSs, flexibility, dependency, performance, and number of occurrences.

b. The manager of Web/Application servers handles information such as performance of the server, number of services running simultaneously, load balancing, and workload.

c. The manager of WS servers handles information such as performance of the server, number of services running simultaneously, load balancing, and workload.

d. The manager of WS containers handles information such as performance, service lifecycle, and number of generated WSDL documents.

e. The manager of the SOAP engine handles information such as performance, number of messages communicated, and number of proxies generated.

The management parameters are saved in the performance package shown in the architecture of Figure 2.

Therefore, the CASE tool enables the administrator to do the following tasks as shown in Figure 7:

- Manage the privileges for different types of developers by creating, removing developers, assigning roles and privileges to developers.
- Manage the composites by adding or removing composites, assigning engines and servers to composites.
- Manage the WS by removing or updating WS, assigning servers to WS.
- Manage the different servers and engines by adding, removing engines and servers.

- Monitor the performance of the WS and the BPEL by tuning the servers, load balancing the services, and controlling the response time, and other BPEL and WS attributes as software.

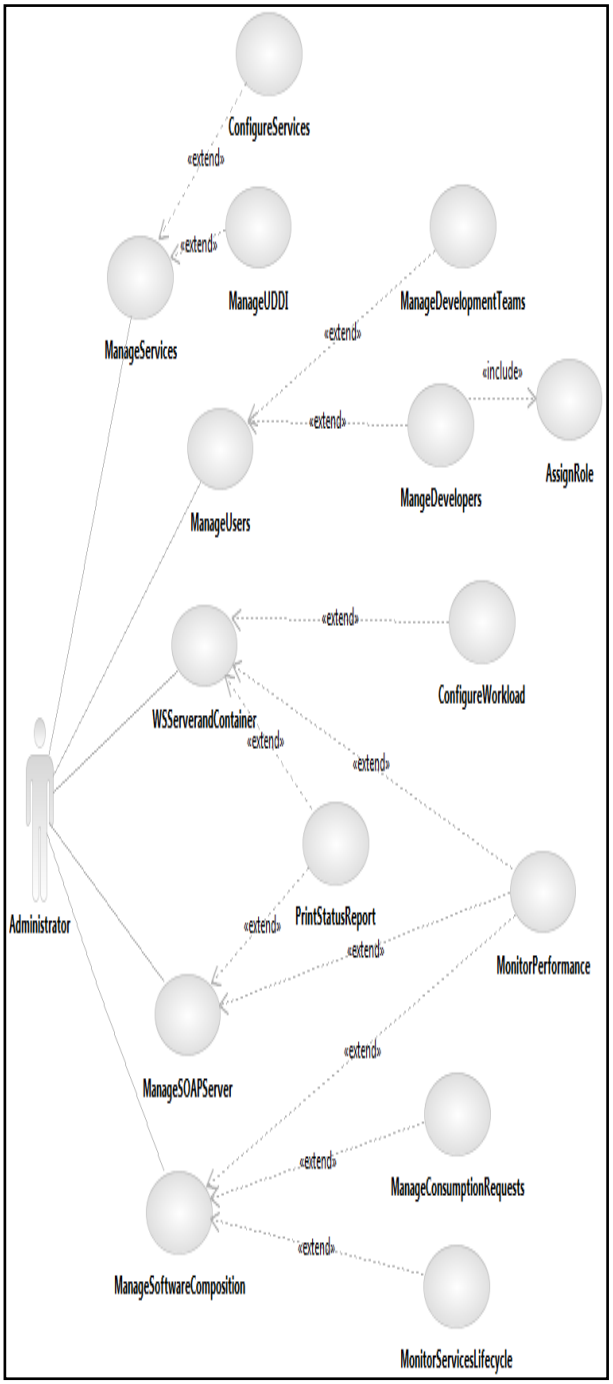

Figure 7. Management use cases

VI. IMPLEMENTATION

The CASE tool is implemented as a web application; using Microsoft Visual C# / ASP.NET and the CASE tool database was created using Microsoft SQL Server. The user may login either as administrator or developer based on privileges authorized. Figure 8 shows the administrator view interface, the admin may select a WS to view more details such as binding part, version and failure impact.

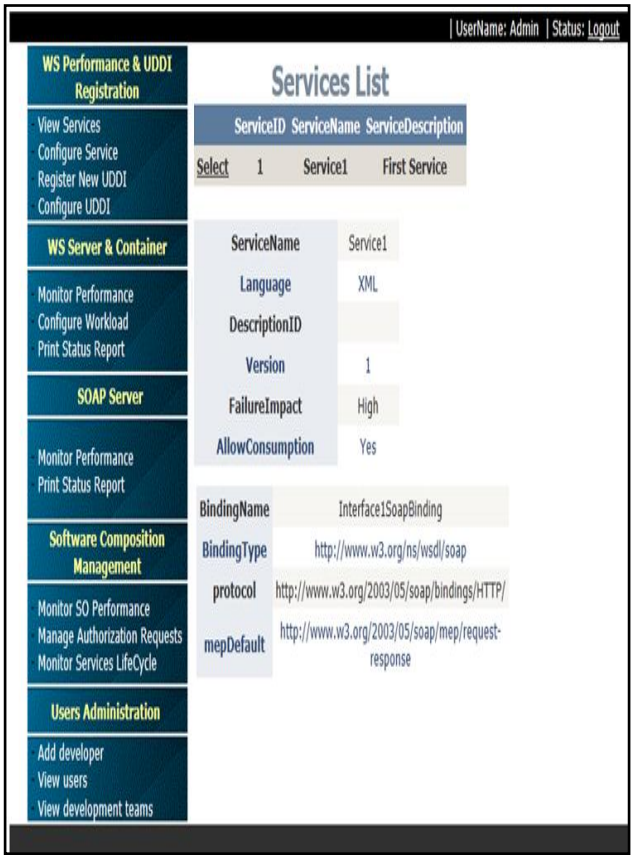

Figure 8. Administrator view/View services interface

Figure 9 shows the developer view services interface, in this view the developer can get a list of registered WSs that may be used in software compositions.

Figure 9. Developer view

VII. RELATED WORK

Although WSs are critical assets for any enterprise willing to redesign its IT infrastructure with SOA, only the development part is considered in some development environments such as IBM Websphere. Few work concern with WSs management. In their work on the extended SOA [9], [10], the authors have added a layer to SOA dealing with management perspectives. In [9], the

authors have summarized many the efforts of [3], [4], [5], [6], [7], [11] in dealing with service management and monitoring. These efforts have set the principles for the monitoring and management.

The approach presented here is an extension of the aforementioned work. It mainly aims at specifying the CASE tool including its architecture design with well-known modeling language that is UML.

This paper presents an architecture, a specification, and an implementation of a CASE tool (with UML notations) for not only WSs stack management and monitoring, but also for SOSE. The architecture of the CASE tool is meant to be flexible and scalable through different abstractions.

VIII. CONCLUSION

This work has presented a specification, an architecture, and an implementation of a CASE tool that assists a comprehensive approach to develop and manage WSs and software as composition of WSs with respect to SOA.

The CASE tool is devised to embed tools such as WSDL generator and wrapper, and BPEL engine in order to any of the well-known approaches that are: top-down, bottom-up, and their extensions green-field and meet-in-the-middlle.

The specification has taken into account different perspectives of the WSs stack and its usage in terms of composites.

The architecture design has considered some design decisions, namely abstraction, flexibility, and agility to come up with five inter-related components.

Both CASE tool architecture and specification are expressed with UML notations in order to be readily and promptly implemented.

The repository representing the WSs has been specified in term of class diagrams modeling its properties with two perspectives that are management and monitoring and SOS development.

The CASE tool is expected to assist developers and managers of services and composition with different perspectives.

This CASE tool is readily extensible to capture an exhaustive list of monitoring and management parameters.

The implementation of the CASE tool needs to integrate existing, generation/wrapper tools and engines.

REFERENCES

[1] M. P. Papazoglou, P. Traverso, S. Dustdar, and F. Leymann, “Service-Oriented Computing: A Research Roadmap”, International Journal of Cooperative Repositories, Vol. 17, No. 2, pp. 223–255, 2008.

[2] P. Bianco, R. Kotermanski, and P. Merson, “Evaluating a service-oriented architecture”, Institute, Technical Report CMU/SEI-2007-TR-015, Sep. 2007.

[3] G. Alonso, F. Casati, H. Kuno, and V. Machiraju, Web services concepts, architectures and applications. Berlin: Springer, 2004.

[4] A. Arora, et al. “Web services for management (WSManagement)”, Technical report, Advanced Micro Devices, Dell, Intel, Microsoft Corporation and Sun Microsystems, October 2004.

[5] L. Baresi, and S. Guinea, “Towards dynamic monitoring of WSBPEL processes”, Proceedings of the Third International Conference on Service Oriented Computing, pp. 269–282. Springer, Amsterdam, 2005.

[6] F. Casati, E. Shan, U. Dayal, and M. C. Shan, “Business-oriented management of web services”, Communications of the ACM, Vol. 46 No. 10, pp. 55-60, 2003.

[7] E. M. Maximilien, and M .P. Singh, “Toward autonomic Web services trust and selection”, Proceedings of the 2nd international conference on Service oriented computing, pp. 212–221. ACM Press, New York 2004.

[8] M. P. Papazoglou and W. J. Heuvel, “Web service management: a survey“, IEEE Internet computing, pp. 58-64, Nov-Dec. 2005.

[9] M. P. Papazoglou and W. J. Heuvel, “Service oriented architectures: approaches, technologies and research issues,” The VLDB Journal, Vol. 16, No. 3, pp. 389-415, Jul. 2007.

[10] D. Georgakopoulos and M. P. Papazoglou (Editors), Service-Oriented Computing, The MIT press, London, England, 2009.

[11] H. Skogsrud, B. Benatallah, F. Casati, “Trust-serv: model-driven lifecycle management of trust negotiation policies for Web services”, Proceedings of the 13th international conference on World Wide Web, pp. 53–62. ACM Press, New York, 2004.

[12] Q. Gu, and P. Lago, “Service identification methods: a systematic literature review”, ServiceWave 2010: 37-50

[13] Q. Gu, and P. Lago, “Guiding the selection of service-oriented software engineering methodologies”, Service-Oriented Computing and Applications, DOI 10.1007/s11761-011-0080-0, 2011, Vol. 5, No. 4, pp. 203-223

[14] IBM, “Rational Rose”, Internet: [Accessed 1-Jun-2012] www.ibm.com/software/awdtools/developer/rose/

[15] P. Brittenham, “Web services development concepts”, IBM Software Group, 2001.

[16] H. Chen, and He, Keqing “A Method for service-Oriented Personalized Requirements Analysis”, Journal of Software Engineering and Applications, Vol. 1, pp. 59-68, 2010.

[17] N. Dubey, “A Paper Presentation on Software Development Automation by Computer Aided Software Engineering (CASE)”, IJCSI International Journal of Computer Science Issues, Vol. 8, No. 1., 2011.

Youcef Baghdadi received his PhD in 1997 in Computer Science from University of Toulouse I, France. He is now an associate professor with the Department of Computer Science at SQU. His research aims at bridging the gap between IT and business in the areas of SOA, WSs, SOC, and SOSE.

Bashar Alani is currently pursuing master degree in computer science at Sultan Qaboos University under the supervision of Dr. Baghdadi, his interests include software engineering, Web services, and SOA.

Zuhoor Al-Khanjari received her PhD in Software Engineering, 1999 from University of Liverpool, UK in Computer Science. She is now an associate professor and chairperson of the Department of Computer Science at SQU. Her research areas include Software Engineering and e-Learning.