Proposal of a Methodology for Implementing a Service-Oriented Architecture in Distributed Manufacturing Systems

I. Medina, A. Garcia-Dominguez, F. Aguayo, L. Sevilla, and M. Marcos

Citation: AIP Conference Proceedings **1181**, 622 (2009); doi: 10.1063/1.3273683
View online: https://doi.org/10.1063/1.3273683
View Table of Contents: http://aip.scitation.org/toc/apc/1181/1
Published by the American Institute of Physics

Proposal of a Methodology for Implementing a Service-Oriented Architecture in Distributed Manufacturing Systems

I. Medina[a], A. Garcia-Dominguez[a], F. Aguayo[b], L. Sevilla[c] and M. Marcos[d]

[a]*Department of Computer Languages and Systems. University of Cadiz. c/ Chile 1, E-11002, Cadiz, Spain*
[b]*Department of Design Engineering. University of Seville. c/ Virgen de Africa, 7, Seville, Spain.*
[c]*Department of Materials, Manufacturing and Civil Engineering. University of Malaga.. Plaza el Ejido s/n, Malaga, Spain*
[d]*Department of Mechanical Engineering and Industrial Design. University of Cadiz. c/ Chile 1,E-11002, Cadiz, Spain.*

Abstract. As envisioned by Intelligent Manufacturing Systems (IMS), Next Generation Manufacturing Systems (NGMS) will satisfy the needs of an increasingly fast-paced and demanding market by dynamically integrating systems from inside and outside the manufacturing firm itself into a so-called extended enterprise. However, organizing these systems to ensure the maximum flexibility and interoperability with those from other organizations is difficult. Additionally, a defect in the system would have a great impact: it would affect not only its owner, but also its partners. For these reasons, we argue that a service-oriented architecture (SOA) would be a good candidate. It should be designed following a methodology where services play a central role, instead of being an implementation detail. In order for the architecture to be reliable enough as a whole, the methodology will need to help find errors before they arise in a production environment. In this paper we propose using SOA-specific testing techniques, compare some of the existing methodologies and outline several extensions upon one of them to integrate testing techniques.

Keywords: Service-oriented architectures, web services, intelligent manufacturing systems, model-driven engineering, testing.
PACS: 07.05.Bx, 64.60.aq, 89.20.Ff, 89.20.Hh.

INTRODUCTION

Nowadays, manufacturing firms must compete in a market with ever shorter product lifespan and increasingly high expectations of flexibility and quality, at competitive prices. To be successful, they must continually adapt and improve their business processes according to the situation and available resources. However, the centralized software platforms generally used in most companies at the time cannot be changed as quickly as needed. These platforms end up defining the firm's practices, rather than the current market situation.

CP1181, *Third Manufacturing Engineering Society International Conference*
edited by V. J. Segui and M. J. Reig

For this reason, a new approach for the information systems in the so-called *Next Generation Manufacturing Systems* (NGMS) is required. On an abstract level, the need to distribute tasks among several specialized information systems is currently acknowledged [1]. The *holonic enterprise* is one of the most common distributed enterprise protomodels [2]. It consists of a collection of holons: semiautonomous agents which collaborate at several stages.

Once the basic conceptual framework has been established, it needs to be implemented in an information system. Fortunately, a new approach for information system architecture design, which fits this framework, has received considerable attention in the past few years: *service-oriented architectures* (SOA). The major underlying concept is not to build a single integrated system for each business unit or project, but rather develop a collection of *services* which can be reused across the organization or even from external business partners. These services can be integrated in higher-level services which provide their own added value and model entire business processes instead of individual tasks. These architectures are usually implemented using *web services* (WS). Web services help leverage existing tools and technologies and reduce the start-up costs.

This approach and its implementation using WS offer attractive gains on both the technical and business sides, but also bring their own complications. Maximum flexibility is achieved through highly granular services, but deploying each service has an additional cost beyond simply coding the required functionality. For this reason, the best option in the short term is to deploy fewer services. Best results are obtained with a medium service granularity, but this is hard to locate. In addition, there are inherent difficulties in developing any distributed system of this magnitude.

Several SOA-oriented methodologies [3-5] have been proposed to control the complexity of this process and ensure more controllable and repeatable results. Some of these modify existing object-oriented methodologies, while others have been created from the ground up around the concept of a "service", as known in SOA. It can be argued that the latter methodologies could provide the best results, as they operate under a higher level of abstraction and represent in a more direct way the business processes to be implemented.

Whichever methodology we choose, there is one more problem: as services are integrated deeper and made more visible, the organization will rely more on their correct operation and potential damage in case of failure will also increase.

This text proposes extending an existing SOA methodology with the models and techniques required to integrate various testing techniques for service-oriented architectures. Several extensions are suggested in order to help locate defects before they are manifested in the production environment in the manufacturing firms' information systems. The present paper is structured as follows: after introducing the basic concepts behind the ideas exposed in this paper, a comparison of the existing methodologies will be performed and one of them will be selected as a starting point for the new methodology. After proposing several extensions to accommodate testing models, this paper will be concluded by a discussion of related works and results obtained.

FOUNDATIONAL CONCEPTS

In this section some of the basic concepts this work is built upon will be briefly described. First, existing approaches for modeling distributed organizations will be discussed. Later, the core elements of the present proposal will be introduced: service-oriented architectures and model-driven development methodologies.

Virtual Organizations: the Extended Enterprise

No enterprise lives in a *vacuum space*: they must participate in their own *ecosystem* consisting of their suppliers, designers, manufacturers, subcontractors, clients and other competing firms. To be successful, a company needs to add more value to its product than its competitors, and this usually means reacting more quickly to market demands using the available information.

However, companies must now interact beyond the local geographical boundaries they used to work in. Communicating different information systems through the Internet is therefore essential, as well as standardizing on the business practices of each stakeholder and solving various logistics problems. The resulting organization is known as an *extended enterprise* [6]. There is a wide spectrum of models for representing the structure of an extended enterprise. According to IMS Consortium, most of them can be categorized either as bionic, fractal or holonic models [2].

Bionic models are inspired on biological systems. The enterprise consists of a collection of tissues (representing processes, products or services), built by cells which perform various tasks and take and produce genetically coded artifacts (parts and products). Likewise, cells are controlled by secreting enzymes (internal control information) and can influence or be influenced by hormones released to the environment (information of the current situation and other cells).

On the other hand, fractal models are based on the mathematical configurations with the same name. As their mathematical counterparts, these models construct the enterprise from self-similar entities at various levels of abstraction. A new fractal is built upon another when the lower level fractal cannot perform all the required tasks by itself. Conversely, the enterprise's goals become increasingly specific as they descend from the main fractal. Each fractal can reorganize and take decisions by itself up to a certain degree.

Lastly, holonic models trace back to the theories about hierarchical systems proposed by Koestler. Koestler defines a holon as an entity which is at the same time a whole built up from lower level holons, and a part of several higher level holarchies. These holons are both partly independent and partly dependent on holons at the same or immediately higher levels.

These three models follow different approaches, but have in common their view of the enterprise as a dynamic network of agents with a limited degree of autonomy and collaboration with others. When implementing one of these models, a recurring problem is how to establish the required communication channels. Most information systems tend to be deployed according to short-term plans specific to each business unit in the organization. Additionally, it might be too risky to replace existing mission-critical legacy systems. These situations present problems in collaborating with firms

in other countries and taking advantage of the new processes and other advantages that could be obtained.

Service-Oriented Architectures

Normally, enterprise information systems are divided into several layers, which are then deployed across several machines. A database collects all the information, which is handled by a business logic layer and then manipulated by the user through a presentation layer. This sort of architecture has been successfully used to create many large computer systems to this date. It naturally fits with centralized, hierarchical and stable organizations.

Nevertheless, it has its drawbacks. These systems tend to be designed to satisfy short-term needs of parts of the organization. They often do not take into account the mid- and long-term need to change the business practices: the business logic tends to be coupled to the design and implementation. Similarly, they usually are not designed to help collaborate with previously unknown companies. Over time, enterprises end up working around the system to innovate, or lose their capability to react to unexpected situations.

For these reasons, a new approach based on service-oriented architectures has received considerable attention recently. Rather than a new set of technologies, it is a different way to organize information systems [7]. Instead of using rigid and centralized structures, information systems are modeled as a collection of reusable services which can be recombined as required to define new business process or refine the existing ones. The underlying ideas are quite similar to those from the holonic or agent-based models: a lower-level service can be part of several higher-level services or business processes by collaborating with other services while having a certain degree of autonomy. This similarity suggests that it would be appropriate to use service-oriented architectures in distributed manufacturing systems.

The technologies most commonly used currently to implement service-oriented architectures are based on web services. These technologies take advantage of many existing tools and open standards to reduce the technological barriers involved.

A common mistake when adopting a SOA is assuming it only takes acquiring and installing the latest version of the chosen SOA software platform. The only way to reap its full benefits is to apply its basic concepts to define a global vision of all services and processes in the organization, and use it to define a catalog of high-quality services. As this is a non-trivial process, it is important to define methodologies which guide its execution, just as when any other sort of software is developed. We will compare some of the existing alternatives in a later section.

Model-Driven Engineering

One of the problems while developing software is that there is only a very weak link between the high-level models (closer to the way we normally think) and the code which implements them. In practice, modeling languages are only used as communication tools between developers and most models are thrown away after the code or the requirements have been changed. This presents problems for several

common tasks, such as verifying if requirements are met, optimizing a design after initial assumptions are changed, take advantage of the latest technologies or checking if the system meets certain properties.

An emerging perspective on software development known as *model driven engineering* (MDE) and advocated by the Object Management Group (OMG) as *model driven architecture* (MDA) [8] attempts to change this situation. It suggests implementing the system from a succession of increasingly detailed models, which map between each other in various levels of automation. It is argued that this would raise the abstraction level at which systems are implemented, in a similar way to what the high-level programming languages achieved in comparison to their low-level counterparts.

Specifically, OMG's proposal defines three types of models for every system. *Computation-independent models* (CIMs) only describe the business environment, ignoring how it will be later reflected in the information system. *Platform-independent models* (PIMs) indicate how to meet the business requirements through the system, without going into the details of how it will be implemented in terms of a particular software and hardware platform. Lastly, *platform-specific models* (PSM) fill the rest of the gaps, in order to simplify their final translation to code.

SERVICE-ORIENTED ARCHITECTURE METHODOLOGIES

In this section some of the existing methodologies for developing information systems based on service-oriented architectures will be reviewed. Specifications and methodologies which only cover specific parts of the software development process, such as the Business Process Modeling Notation (BPMN) [9] will be discarded. For instance, the latter specification only describes how to model business processes, and not how they should be implemented.

Precedents in Component Based Development

As said above, SOA is not a revolution, but an evolution due to the lessons learned in the field of Software Engineering. There are similarities between SOA and the immediately previous conceptual step: component based development (CBD) [3]. Many definitions exist for "component", but it can be broadly described as a self-sufficient "black box" which provides some service without detailing how it is realized, and which can be integrated with other components. The main difference between SOA and CBD is how those services are used: in CBD, the component becomes an indivisible part of the main program, while in SOA it remains as an independent entity which exchanges messages with other programs. Moreover, SOA's services implement business-level logic, while components may implement lower-level logic.

SOMA Methodology

The *Service Oriented Modeling and Architecture* (SOMA) methodology developed at IBM [4] defines an integrated approach over the SOA development process. SOMA

spans from its conception to its monitoring and maintenance. It is organized as an iterative improvement cycle, which is further divided into several stages.

First, a business model is defined, along with a set of templates for each of the possible integration solutions. Next, the services to be included in the architecture will be collected, using information from several sources: the organization's goals, a conceptual model of the environment, and existing information systems.

Later, all services will be re-factored, rationalized and specified as a part of a coherent architecture. As a large number of services might have been identified, a subset containing those with the highest returns on investment will be selected, and the rest will be postponed. Finally, the selected services will be implemented, debugged, deployed and monitored.

The methodology has been implemented as a set of extensions to the Rational Unified Process (RUP), and uses a slightly extended form of the Unified Modeling Language (UML) [10]. This implies several advantages and disadvantages: though it has been successfully used in several projects and is based on a well-known process, it is a complex methodology which requires producing a large volume of documentation and using many tools. For this reason, it may not be the most adequate option for medium and small manufacturing firms, with fewer resources to dedicate to modeling. A more lightweight approach would be more effective in these cases.

SOD-M Methodology

The *Service Oriented Development Method* (SOD-M) [5,11] is also model-driven and centered on services. It covers the three OMG MDA viewpoints over the business aspects (CIM) and the information system (PIM and PSM). The business viewpoint includes models of the value exchanges between the organization and its environment [12] and descriptions of the existing business processes as UML activity diagrams. These diagrams are similar to the well-known flow diagrams.

A set of use case models for the information system is defined at the PIM level. Later on, these models are extended into service process models, which detail the activities required for their completion and their relationships with other processes. These activities will be then distributed among the stakeholders in the service composition models. Both of these two models use UML activity diagrams.

At the PSM level, the extended service composition models specify what activities from the original service composition models will be exposed as web services. Interfaces will be defined for each of these services.

This methodology is simpler than SOMA, and some of the mappings between the different models are partially automated [11], unlike other approaches, such as [3]. A diagram of the relationships between the different models is shown in figure 1. SOD-M uses simpler notations, helping clients and developers communicate. However, unlike SOMA, SOD-M does not include all the steps in the development process. Specifically, it does not integrate software testing activities.

Nevertheless, it can be considered as a good basis for an integrated methodology for the development of service-oriented information systems, where the current business practices of the organization can be reflected.

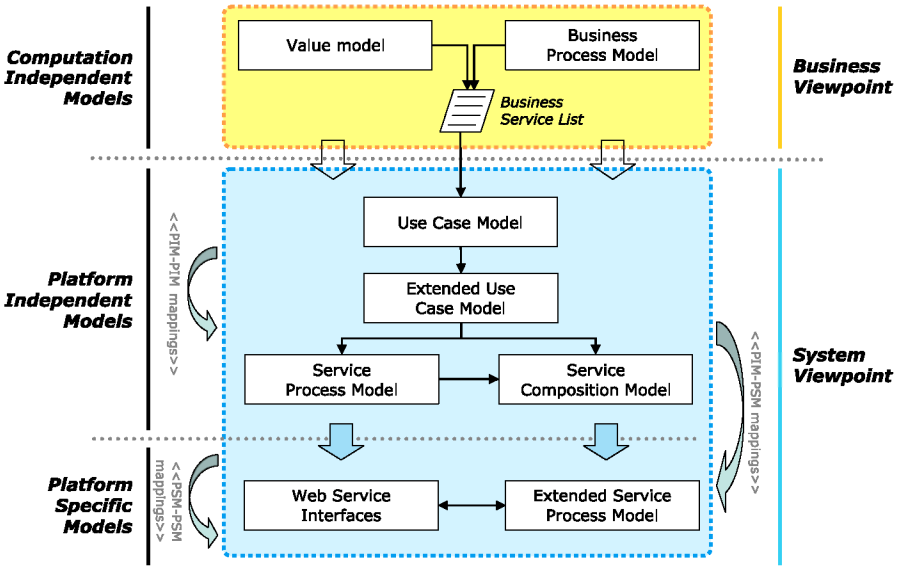

FIGURE 1. Diagram of the models used in SOD-M

INTEGRATION OF TESTING MODELS INTO SOD-M

In the previous section some of the existing methodologies were reviewed, and SOD-M was selected as a valid starting point for defining a lightweight integrated SOA methodology. However, it lacked an important activity: testing the software. In this section several extensions to SOD-M in order to cover that gap are proposed.

Test Categories

Broadly speaking, there are six kinds of tests for information systems [13]: installation, acceptance, system, function, integration and module (also known as unit) testing. Installation and module testing do not require a different approach in the context of SOA, so they will not be discussed any further in the present paper.

System tests in SOA would test the whole architecture, that is, the ecosystem created from all the exposed services and their consumers. They could check whether our system would at some point invalidate some of its performance constraints, such as the expected number of transactions processed per minute, logged in users or maximum data throughput, or fail to enforce some access restriction rule, among others. These properties are usually part of its Service Level Agreement (SLA).

Function tests, on the other hand, only consider a single business process, which generally composes a collection of smaller web services into a more comprehensive single web service (using the Web Service Business Process Execution Language or WS-BPEL, for instance). Testing must ensure that the desired functionality has been implemented and that other restrictions have been met. These restrictions are usually those from the system tests, but applied at a lower level.

Integration tests are limited to a single web service, usually implemented by assembling multiple modules. The tests to be performed are again both similar and more specific than those in the upper levels, except for the fact that in most cases, the elements under test will be written using general purpose programming languages, rather than using domain specific languages to compose services, such as WS-BPEL.

Proposed Extensions

Acceptance tests can be assisted by the traceability relationships between the models in the CIM and PIM levels. If we follow the relationships from each business process activity up to the web service level and verify that these have been implemented correctly, we will have a high degree of confidence that the requirements have been met. Therefore, there is no need to change the model in this case, but rather the tools that manipulate it.

System testing could use an extended version of the service process PIMs which would include assertions about the expected service level (in terms of performance, response time, etc.) and access restrictions. These annotations would be performed at a global level and then propagated through the structure of all business process to each individual activity. Every activity would need to aggregate the annotations from each business process it participated in, producing information for the PSMs. Test cases to verify whether some of the requirements were not met at some point in time could be derived from these specific requirements.

Figure 2 shows a sample service process model for attending to a request from a client. The model has been decorated with several stereotyped comments, indicating the minimal number of transactions per second which should be processed and placing a time limit on each of them. Conditional branches include probability estimations for each branch. From the information manually specified by the designer (inside comments with colored backgrounds) new information could be automatically derived (comments with clear backgrounds). For example: if the probability of accepting the request is $p=0.8$, each of the two concurrent execution paths should be able to process $5p=4$ transactions per second.

There are two options for performing the function and integration tests: either generating and running test cases with inputs and their expected outputs, or checking several properties on the implemented code without running it. In both, models listing what test cases to run or what checks to make will be useful. Compositions in their service composition models and activities to be exposed as web services in the extended service composition models could include sets of logical conditions [15] defining the expected relationships between inputs, outputs, and changes on the stored information. SOD-M by itself does not include models to represent the information stored in the system, but its authors have already taken it into account, integrating SOD-M with the Web information system development methodology MIDAS [5]. MIDAS includes a conceptual data model at the PIM level.

Figure 3 illustrates a sample service composition model which has been derived from the previous business process model. The conditions to be met at the beginning and end of the execution of the composition have been marked with the standard stereotypes <<*precondition*>> and <<*postcondition*>>. Likewise, the conditions to

be met before and after the "Create receipt" web service has been invoked are listed with the *<<localPrecondition>>* and *<<localPostcondition>>* stereotypes. The conditions for this web service specify that the request should initially be in an open state, accepted, not empty, and should not already have a receipt created for it. After it has been invoked, a new receipt with the correct articles, prices and total sum should have been created and sent to the "Perform payment" activity for later processing.

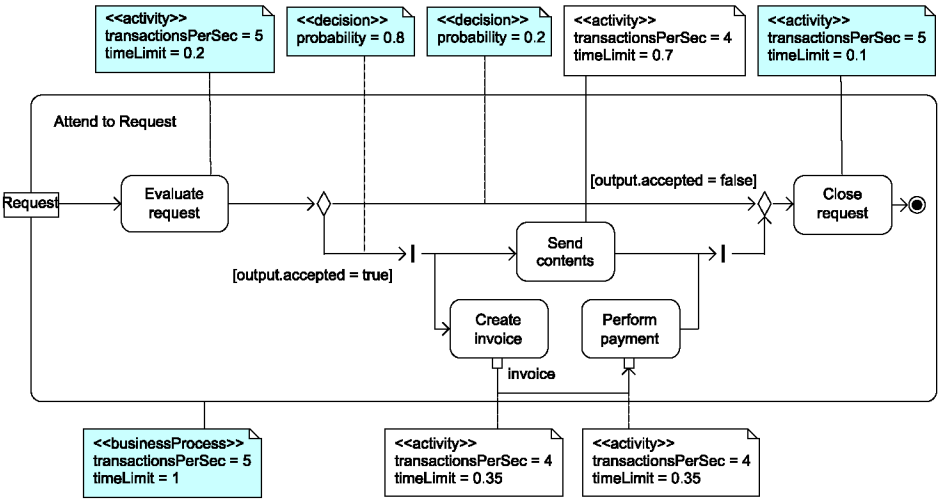

FIGURE 2. Service process model decorated with manual and automatic service level expectations

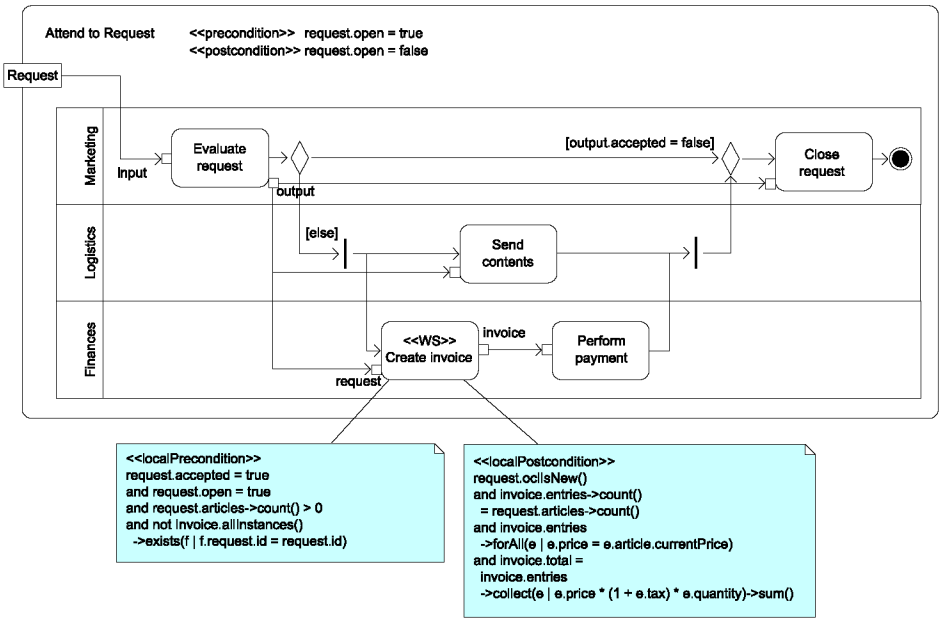

FIGURE 3. Sample service composition model annotated with OCL assertions

Once the behavior of all service processes and web services has been specified, the specifications can be used to integrate various testing techniques. For instance, a cause-effect graph [13, 16] from which test cases could be semiautomatically extracted could be derived, among other testing oriented models, such as those in [17].

Only generating a large number of test cases is usually not enough: the test suite must meet a minimum level of quality. Parts of this evaluation can be performed over the specification (that is, our models), and other parts need to be done by running the implemented code [18]. For instance, [19] helps finding potential coding mistakes that the test cases would not detect, and [20] identifies properties which can differ from those expected from the composition. There are other options, such as calculating the percentage of all instructions which were run by a set of test cases, for instance.

Ultimately, after running the tests over the final executable versions of the services and compositions and locating defects and gaps in the test suites with them, the intermediate models could be refined to integrate our new knowledge. This knowledge could then be ported to another platform or serve as the base for the new tests to be performed after the next change, making sure everything that worked before still works as expected.

CONCLUSIONS AND FUTURE WORKS

To be competitive, manufacturing firms must integrate the latest technologies in manufacturing processes and revise their business practices according to market demand. This requires the organization to change from a single centralized and hierarchical entity to a collection of dynamically interrelated elements, which integrate the suppliers, clients, designers, subcontractors and other stakeholders in a so-called extended enterprise. The importance of this fact can be seen in the emergence of distributed enterprise models in the field of Manufacturing Engineering, such as the holonic, fractal and bionic enterprises.

It is argued in this paper that such an extended enterprise should have an information system fitting that vision. It would follow a service-oriented architecture, where the whole system is defined as an ecosystem of services produced and consumed by different parties, which can be freely recombined to change the current business practices and integrate external systems. However, it needs to be done carefully, as it involves the whole organization, and a defect in a widely reused service could have grave consequences. Therefore, it is required to follow a well-defined methodology to simplify communication between developers and users and ensure the development of an architecture whose services meet the defined requirements.

Future work will start by integrating the techniques which provide the most immediate benefits. These would be the modifications to the service process models to integrate requirements over response time, expected performance, and the probability distributions for the conditional branches. They would require extending the metamodels included in the SOD-M methodology, and implementing of the automatic inference logic in the selected modeling tools.

Later on, a notation ([15] is a first candidate) will be selected for specifying the behavior of each service composition model according to the relationships between their inputs and outputs and the changes over the stored information. This notation

will then be adapted to generate test cases and perform various checks against implementations of those compositions using domain specific languages such as WS-BPEL [14]. The previous adaptation of the notation to WS-BPEL will be then extended to individual web services.

Finally, test coverage evaluation techniques will be added into the methodology, to provide measures of the quality of the tests run. Several coverage criteria, based on the code of the program, could be combined with more advanced tools, such as [20], which could consider the coverage of the logic itself, rather than only its structure.

REFERENCES

1. M. Marcos, F. Aguayo, M. Sánchez Carrilero, L. Sevilla and J.R. Lama, "Toward the Next Generation of Manufacturing Systems. Frabiho: a Synthesis Model for Distributed Manufacturing", in *Proceedings of the First I*proms Virtual Conference*, Elsevier, 2005, pp. 35-40.
2. A. Tharumarajah, A. Wells and L. Nemes, "Comparison of emerging manufacturing concepts", in *Proceedings of the 1998 IEEE International Conference on Systems, Man, and Cybernetics*, California, USA, 1998, pp. 325-331.
3. Z. Stojanović, "A Method for Component-Based and Service-Oriented Software Systems Engineering". Ph. D. Thesis, Delft University of Technology, 2005.
4. S.G. A. Arsanjani and A. Allam, "SOMA: a method for developing service-oriented solutions", *IBM Systems Journal* **47**, 377-396, 2008.
5. M.V. de Castro, "Aproximación MDA para el desarrollo orientado a servicios de sistemas de información web: del modelo de negocio al modelo de composición de servicios web", Ph. D. Thesis, University Rey Juan Carlos, 2007.
6. J. Browne, I. Hunt and J. Zhang, "The Extended Enterprise (EE)", in *Intelligent Systems for Manufacturing: Multi Agent Systems and Virtual Organizations*, edited by L.M. Camarinha-Matos, H. Afsarmanes and V. Merik, Kluwer Academic Publishers, London, 1998, pp. 3-30.
7. T. Erl, *SOA: Principles of Service Design*, Indiana, USA: Prentice Hall, 2008, ISBN 0-13-234482-3.
8. Object Management Group, "MDA Guide version 1.0.1", June 2003. See: http://www.omg.org/mda/.
9. Object Management Group, "Business Process Modeling Notation 1.2", January 2009. See: http://www.omg.org/spec/BPMN/1.2/.
10. Object Management Group, "Unified Modeling Language 2.1.2", November 2007. See: http://www.omg.org/technology/documents/modeling_spec_catalog.htm.
11. J.M. Vara Mesa, E. Marcos and M.V. de Castro, "Obteniendo modelos de sistemas información a partir de modelos de negocios de alto nivel: un enfoque dirigido por modelos", in *Proc. IV Jornadas Científico-Técnicas en Servicios Web y SOA*, Seville, Spain, 2008, pp. 15-28.
12. J. Gordijn, "Value-based requirements engineering: Exploring Innovative e-Commerce Ideas", Ph. D. Thesis, Vrije Universiteit, 2002.
13. G.J. Myers, *The Art of Software Testing*, John Wiley & Sons, second edition (2004), ISBN 0471469122.
14. OASIS, "WS-BPEL 2.0 Standard", April 2007. See: http://docs.oasis-open.org/wsbpel/2.0/OS/wsbpel-v2.0-OS.html.
15. Object Management Group, "Object Constraint Language Specification 2.0", May 2006. See: http://www.omg.org/technology/documents/formal/ocl.htm.
16. A. Paradkar, M.A. Vouk and K.C. Tai, "Specification-based testing using cause-effect graphs", in *Annals of Software Engineering* **4**, 133-157, January 1997.
17. Y. Zheng, J. Zhou and P. Krause, "An Automatic Test Case Generation Framework for Web Services", *Journal of Software* **2**, 64-77, September 2007.
18. H. Zhu, P. Hall and J. May, "Software Unit Test Coverage and Adequacy", *ACM Computing Surveys* **29**, 366–427, December 1997.
19. J. J. Domínguez Jiménez, I. Medina Bulo and A. Estero Botaro, "A framework for mutant genetic generation for WS-BPEL", in *Proceedings of the 35th Conference on Current Trends in Theory and Practice of Computer Science (SOFSEM 2008)*, Spindleruv Mlýn, Czech Republic, 2008.
20. M. Palomo-Duarte, A. García Domínguez and I. Medina-Bulo, "Improving Takuan to analyze a meta-search engine WS-BPEL composition", in *Proceedings of the 4th IEEE International Symposium on Service-Oriented System Engineering*, Jhongli, Taiwan, 2008.