Assessment Scheme-based Service Selection for SOC-based Applications[1]

Martín Garriga[1,3], Andres Flores[1,3], Alejandra Cechich[1], and Alejandro Zunino[2,3]

1 GIISCo Research Group, Facultad de Informática, Universidad Nacional del Comahue, Neuquén, Argentina. [aflores,acechich]@uncoma.edu.ar,
2 ISISTAN Research Institute, UNICEN,
Tandil, Argentina, azunino@isistan.unicen.edu.ar
3 CONICET (National Scientific and Technical Research Council), Argentina.

Abstract. Service-Oriented Computing promotes building applications by consuming reusable services. However, facing the selection of adequate services for a specific application still is a major challenge. Even with a reduced set of candidate services, the assessment effort could be overwhelming. On previous works we have presented a novel approach to assist developers on discovery, selection and integration of services, specially focusing in the selection method, which is based on a comprehensive scheme for services' interfaces compatibility. The scheme is also complemented by a framework based on black-box testing to verify compatibility on the expected behavior of a candidate service. This paper analyzes the selection method through a series of case studies, which are designed to show the scheme's potential on determining the best choice of a service among a set of candidates.

Keywords: Service oriented Computing, Component-based Software Engineering, Web Services

1 Introduction

Service-Oriented Computing (SOC) promotes building distributed applications in heterogeneous environments [1]. Service-oriented applications are developed by reusing existing third-party components or services that are invoked through specialized protocols. The SOC paradigm has been widely adopted by using the Web Services technology [2], which leads to a concrete descentralization of bussiness processes and a low investment of new technologies and execution platforms. However, the efficient reuse of existing Web Services is still a major challenge. On one side, searching for candidate services on the Web implies a manual task yet, mainly exploring web catalogs usually showing poorly relevant information. On the other side, the result of a prosperous search requires skillful developers to deduce from the set of candidates, the most appropriate service to be selected for the subsequent integration tasks. Even with a reduced set of services, the required assessment effort could be overwhelming. Not only functional and non-functional

[1] This work is supported by projects: ANPCyT–PAE-PICT 2007-02312 and UNCo–IEUCSoft (04-E072

properties must be explored on candidates, but also the required adaptations for a correct integration allowing client applications to consume services while enabling loose coupling for maintainability.

In order to ease the development of SOC-based applications we presented on a previous work [3] a proposal for *discovery*, *selection* and *integration* of services, which is based on two recent approaches concerned on development and maintainability. The first approach, called EasySOC [4], provides specific semi-automated methods for both *discovery* and *integration* of services. The second approach, was initially developed as a solution for substitutability of component-based systems [5]. This approach supplies a method for *selection* of the most appropriate third-party candidate component. Since web services involve a special case of software component [6][7][8], few initial adjustments were required to apply this selection method on the context of service-oriented applications.

Particularly, this paper presents an extention of the *selection method* where a comprehensive scheme has been defined for assessing interfaces of candidate services according to requirements of internal components from a SOC-based application. The scheme allows to characterize the matchmaking process through a series of syntactic compatibility cases conveying not only the usual programming standards (e.g. names on operations and parameters), but mainly differentiating strong and potential similarity cases. The assessment process thus may produce an automatic identification of certain similarity cases to then giving the chance to improve the compatibility result by solving mismatch cases or better low equivalence results. The assessment scheme is also complemented by a framework based on black-box testing to explore the required behavior for candidate services, where the goal is to fulfill the observability testing metric [9] that identifies a component operational behavior by analyzing data transformations (input/output), which helps to understand the functional mapping performed by a component and therefore its behavior. Hence, a potential compatibility of a candidate service could be exposed – as we analyzed on a previous work [3] and was also discussed in [5][10].

Both approaches are supported by semi-automatic tools, named EasySOCPlugin and TestOOJ respectively that have been conveniently integrated, to support the whole new approach and validating the ideas proposed in this paper.

The paper is organized as follows. Section 2 presents an overview of the whole process for SOC-based application development. Section 3 gives details of the Assessment Scheme of the Selection Method. Section 4 presents a series of case studies. Conclusions and future work are presented afterwards.

2 Process for SOC-based Application Development

During development of a service-oriented application, a developer may decide to implement specific parts of a system in the form of in-house components. However, the decision could also involve the acquisition of third-party components, which in turn could be solved with the connection to web services. Figure 1 depicts our proposal intended to assist developers in the process of *discovery*, *selection* and *integration* of web services, which is briefly described as follows:

The first phase related to web services *discovery* is achieved by applying a combination of text mining and machine learning techniques. A simple input specification (in the form of a required interface I_R) is processed to form a specialized query sentence, and a search method, called WSQBE [4], returns a short list of candidate services through a mechanism for search space reduction. The second phase for services *selection* is described in more detail below according to Figure 2. The third phase related to Web services *integration* is based on an model intended to allow a client in-house component (*C*) being not strongly coupled to a service's interface (I_S) and also unaware of physical invocation aspects, e.g., interaction protocols, datatype formats, location, etc. Therefore, physical details for invoking services are deployed as a separate layer, through a service adapter (A_S) and a service proxy (P_S), which is placed in between to abstract client components (*C*) from changes on services' interfaces. Thus, client applications are able to operate with different interfaces by altering the intermediate layer, while the code implementing their in-house components remains untouched [4].

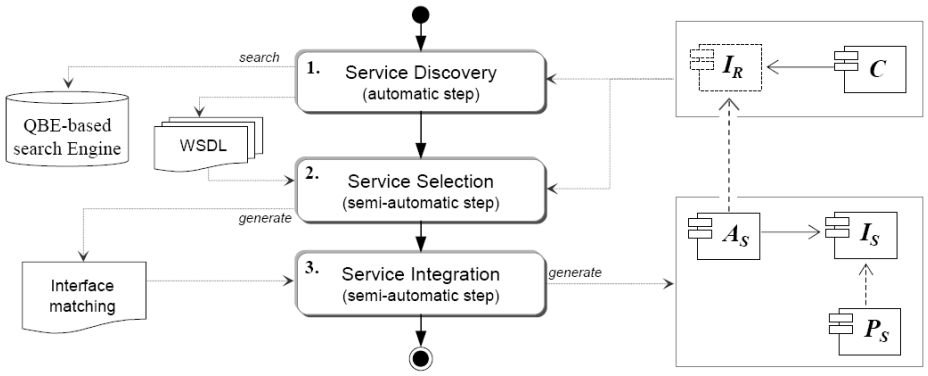


Fig. 1. Process for SOC-based Application Development

The *selection method* provides two main assessment procedures: an Interface Compatibility analysis and a Behavioral Compatibility evaluation, as shown in Figure 2. The Interface Compatibility evaluation is based on a comprehensive Assessment Scheme to recognize strong and potential matchings from a required interface (I_R) and the interface provided by candidate services (I_S). The outcome of this step is a Syntactic Matching List where each operation from I_R may have a correspondence with one or more operations from I_S [5]. Since this step is the main focus of this paper, details are given in Section 3.

The Behavioral Compatibility evaluation is intended to complement the previous assessment, where a Behavioral Test Suite (TS) is built to represent behavioral aspects from a third-party service, with required interface I_R. For this evaluation, the Syntactic Matching list produced in the previous step is processed, and a set of wrappers (adapters) is generated to allow executing the TS against the candidate service (through its provided interface I_S) to evaluate the achieved behavior compatibility [5].

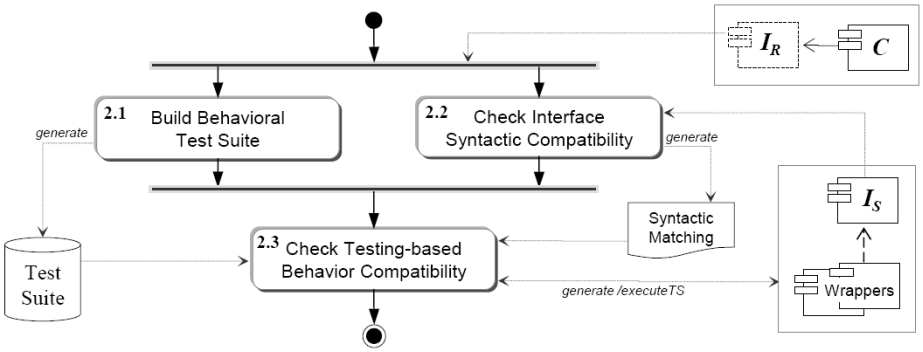


Fig. 2. Selection Method

Next sections provide detailed information particularly related to the Interface Compatibility step. A case study will be used to illustrate the usefulness of the Assessment Scheme into the Selection Method.

2.1 Case Study

Let us suppose the development of a communication tool for exchanging instant messages with contacts from a user's contact list. We have specified the behavior of the required service in the form of operations defined into a Java interface I_R, named `ChatIF`. Figure 2(a) shows the required interface `ChatIF`, which includes a complex type named `Content`.

By running the first phase of the process, a set of web services called OMS (Online Messenger Service) has been discovered at *http://www.nims.nl/*. Particularly we are interested on two of those services: `OMS2` and `OMS2_Simple`. The former (*http://www.nims.nl/soap/oms2.wsdl*) provides an interface I_{S1} comprised of 38 operations, and the most relevant ones are shown in Figure 2(b), where another complex type named `Message` is used for enclosing the contents to be exchanged. The latter (*http://www.nims.nl/soap/oms2_simple.wsdl*), whose interface I_{S2} is shown in Figure 2(c), uses the `String` type for the operations return, instead of any other type (built-in or complex).

3 Interface Compatibility Analysis

The Selection Method corresponds to the second phase of the whole process for SOC-based application development. Two main evaluations are applied on candidate services, from which a concrete recommendation concerning the most appropriate service is achieved. The final evaluation procedure (*step 2.3*) takes the set of candidate services to be put under test with the purpose to discover compatibility with respect to the expected behavior for the client application. Nevertheless, such final evaluation requires a previous assessment at a syntactic level on Interface Compatibility (*step 2.2*), which may provide useful preliminary information to help developers gain knowledge on several aspects.

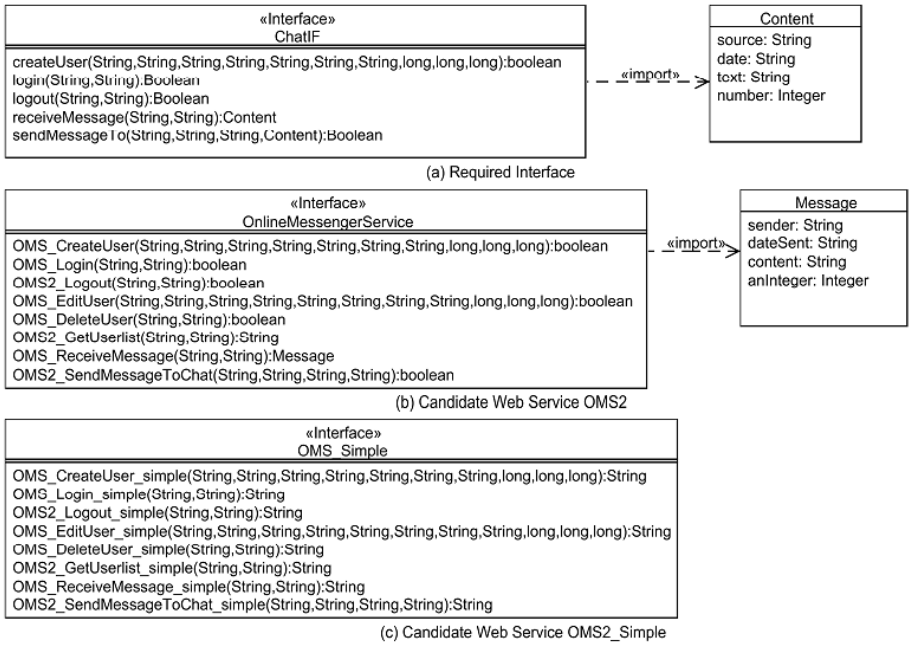


Fig. 3. Instant Messenger Application – Chat

Particularly, the Interface Compatibility analysis is comprised of a practical scheme of two parts: automatic matching cases and semi-automatic potential matchings, to analyze operations from the interface I_S (of a candidate service *S*), respect to the required interface I_R.The outcome of this step may avoid early discarding a candidate service upon simple mismatches but also preventing from a serious incompatibility. In addition, helpful information about the adaptation effort of a candidate service may take shape for a positive integration into the consumer application.

3.1. Assessment Scheme

Table 1 presents the Assessment Scheme that is comprised of four levels to define different syntactic constraints for a pair of corresponding operations. Constraints are based on individual conditions, summarized in Table 2, according to the elements of an operation' signature (return, name, parameter, exception). Types on operations from I_S should have at least as much precision as types on I_R. However, the String type is a special case, being considered as a *wildcard* type since it is generally used in practice to allocate different kinds of data. *Parameters* (`P`) and *return type* (`R`) are the most significant signature elements of the scheme. To consider an initial strong compatibility result, a criterion of "*no inclusion*" has been defined for conditions `R3` and `P4` that are evaluated in the Automatic part of the scheme as incompatibilities (treated as conditions `R0` and `P0` respectively). Therefore, those weakest compatibility cases (`R3` and `P4`) are managed under the Semi-Automatic part of the scheme – e.g., operation `sendMessageTo` of `ChatIF` in Table 1.

Table 1. Assessment Scheme: Automatic Match and Semi-Automatic Mismatch Solving

Level	Part	Constraints
■ Exact Match	Auto (1 case)	Two operations must have identical signatures. (four identical conditions): [R1,N1,P1,E1]
■ Near Exact Match	Auto (13 cases)	Three or two identical conditions. The remaining might be second conditions: (R2/N2/P2/E2). Exceptional cases: three identical conditions with a remaining third condition (N3/P3/E3)
	Example: operation `logout` of `ChatIF` has *near-exact_2* match to `OMS2_Logout` of `OMS2` with a substring equivalence for the operation name ("logout"): [R1,N2,P1,E1]	
	Semi-A (1 case)	Three identical conditions with the return that may have a no equivalent complex type or lost precision: [R3,N1,P1,E1]
■ Soft Match	Auto (26 cases)	Similar to the previous level, but only two identical conditions. Previous exceptional cases may occur with lower equivalence conditions.
	Semi-A (13 cases)	Two identical conditions, similar to automatic scheme. Either return or parameter (not both) with a nonequivalent complex type or lost precision (R3/P4).
	Example: operation `sendMessageTo` of `ChatIF` could match operation `OMS2_SendMessageToChat`. However, the first operation includes a parameter of complex type (`Content`) without a match into the other operation that has only String parameters (initially evaluated as P0). This can be re-evaluated considering that the wildcard type String might contain a chain of all fields from the complex type – i.e. an equivalence *soft_25*: [R1,N2,P4,E1].	
■ Near Soft Match	Auto (14 cases)	There cannot be two identical conditions, i.e. all conditions can be relaxed simultaneously.
	Semi-A (40 cases)	Either two identical conditions with the condition P4 or relaxing all conditions simultaneously.

The Assessment Scheme in Table 1 is able to recognize 108 cases for Interface Compatibility (where each part is comprised of 54 cases), from the combination of individual conditions (classified into the four levels of compatibility).

For complex data types their comprising fields must be equivalent one-to-one with fields from a complex type counterpart. For example, `receiveNextMessage` of `ChatIF` and `OMS_ReceiveMessage` of `OMS2` have a complex type as a return (`Content` and `Message` respectively), which are equivalent (R2) because their fields are equivalent one-to-one. Thus, these operations have a *near_exact_12* match, since they also coincide on parameters and exceptions (P1,E1); with a substring equivalence on their names (N2) – common words "*receive*" and "*message*".

When certain mismatch cases are detected for the interface I_R, a developer may outline a likely solution with the support of context information from the application's business domain. We have identified specific cases in which a concrete compatibility can be set up providing a semi-automatic mechanism to ease this procedure. An example is given in Table 1 with the operation `sendMessageTo` of `ChatIF`.

Table 2. Syntactic Operation Matching Conditions for Interface Compatibility

Return Type	R0: Not compatible	R1: Equal return type
	R2: Equivalent return type (subtyping, Strings or Complex types)	R3: Not equivalent complex types or lost precision
Name		N1: Equal operation name
	N2: Equivalent operation name (substring)	N3: Operation name ignored
Parameters	P0: Not Compatible	P1: Equal amount, type and order for parameters
	P2: Equal amount and type for parameters	P3: Equal amount and type at least equivalent (including subtyping, Strings or Complex types) for some parameters into the list
	P4: Not equivalent complex types or lost precision	
Excep-tions	E0: Not compatible	E1: Equal amount, type, and order for exceptions
	E2: Equal amount and type for exceptions into the list.	E3: If non-empty original's exception list, then non-empty candidate's list (no matter the type).

The second part of the Scheme is not only intended to assist on solving mismatch cases, but also to allow a developer to "force" certain correspondences even when an automatic match was identified. For a specific operation $op_R \in I_R$, there could be another correspondence that better fit for the application's context. The developer is enabled to make such prioritization, which then is considered in first order for the processing on the Selection Method's subsequent step (see Section 2).

The final outcome of the Interface Compatibility step is a matching list characterizing each correspondence according to the four levels of the Assessment Scheme, named *Interface Matching List*. For each operation $op_R \in I_R$, a list of compatible operations from I_S is shaped. For example, let be I_R with three operations op_{Ri}, $1 \leq i \leq 3$, and I_S with five operations op_{Sj}, $1 \leq j \leq 5$. The matching list might result as follows: { (op_{R1}, {op_{S1}, op_{S5}}), (op_{R2}, {op_{S2}, op_{S4}}), (op_{R3}, {op_{S3}}) }.

Each compatibility case represents a specific numeric value in the Assesment Scheme. For example, the value of *exact* equivalence is 4. Therefore, a totalized value could be determined to synthetize the *degree* of Interface Compatibility between a required interface I_R and a candidate interface I_S (from a service *S*). Only the higher compatibility level for each operation is considered to calculate that value, named *Syntactic Distance.* The corresponding formula is shown in (1).

$$syntDist(I_R,I_S) = \frac{\sum_{i=1}^{N} Min(op_{Ri}, MapComp(I_R,I_S))}{N * 4} - 1 \qquad (1)$$

where *N* is the interface's size of I_R, and *MapComp* are the values for the compatibility cases found for operation op_{Ri}.

If all operations in the *Interface Matching List* presents an *exact* equivalence, the *Syntactic Distance* between I_R and I_S is zero. This iniatially means that I_R is included into I_S, though I_S may have additional operations.

The success on the precision achieved during the Interface Compatibility step is essential to reduce the computation effort for the subsequent step of behavior evaluation (see Section 2). This is the main reason for the definition of the whole Assessment Scheme, in which different design and programming heuristics have been applied, mostly from a practical experience perspective.

4 Case Studies

This section shows the evaluation's results for the example presented in Section 2.1. Then another case study is briefly described.

4.1 Instant Messenger – Chat

Table 3 shows the automatic matching results for `ChatIF` and service `OMS2`, where a mismatch is identified for operation `sendMessageTo` of `ChatIF` (depicted with a gray cell) for which a semi-automatic solution could be set up by a *soft_25* (`R1,N2,P4,E1`) match to operation `OMS2_SendMessageToChat` of `OMS2`. The rest of the `ChatIF` interface has found a match. For example operation `createUser` has a *near-exact_2* match to operation `OMS_CreateUser` (due to the substring equivalence). Operations `login` and `logout` obtained similar result by a *near-exact_2* match to alike operations, and four *near-exact_7* matches to other operations.

Table 3. Automatic Interface Compatibility between `ChatIF` and `OMS2`

ChatIF	OMS2
boolean createUser(String, String,String,String,String, String,String,long,long,long)	[n_exact_2, boolean OMS_CreateUser (String, String, String,String,String,String,String,long,long,long), R1, N2, P1, E1]
boolean sendMessageTo (String,String, String,Content)	
Content receiveNextMessage (String, String)	[n_exact_12,Message OMS_ReceiveMessage (String, String,), R2, N2, P1, E1]
boolean logout(String, String)	[n_exact_2,boolean OMS2_Logout(String,String), R1, N2, P1, E1]
boolean login(String, String)	[n_exact_2, boolean OMS_Login(String, String), R1, N2, P1, E1]

As no automatic matching has been found for `ChatIF` and `OMS2Simple`, the mismatches have been solved in the semi-automatic step, by the notion of the String type as a *wildcard* type (see Section 3.1).

At this point, the *Interface Matching List* for both candidate services is available. Thus, the *syntactic distance* could be used to determine which of them is better to continue with the Behavioral Compatibility (*step 2.3*). Table 5 summarizes the best values found for each candidate service and each operation in `ChatIF`.

The *syntactic distance* between `ChatIF` and `OMS2` is 29/20-1 = 0,45 according to formula (1), and considering `OMS2_Simple` the *syntactic distance* is 40/20-1 = 1. Because the lower value is better, the suggested candidate service is `OMS2`.

Table 4. Interface Compatibility Summary for `ChatIF`, `OMS2` and `OMS2Simple`

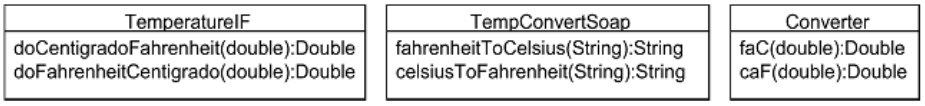
ChatIF Operations	OMS2 Best Value*	OMS2_Simple Best Value*
createUser	5	6
sendMessageTo	8	11
receiveNextMessage	6	7
logout	5	8
login	5	8
Total	29	40
Syntactic Distance	0,45	1

* Total Best Value 20 (based on `ChatIF` size)

4.2 Weather System

This case study is a system in which it is required to provide temperature information on both Celsius and Fahrenheit scales. A required interface I_R has been defined in the Java format, named `TemperatureIF`, which is shown in Figure 4(a). Candidate web services are named `TempConvert`[2] and `Converter`[3], whose interfaces I_{S1} and I_{S2} are shown in Figure 4(b) and 4(c) respectively.

The automatic Interface Matching between `TemperatureIF` and service `TempConvert`, reveals that all operations from `TemperatureIF` have found a match. Both operations from `TemperatureIF` obtained similar result by two matches to both operations of `TempConvert` service. The `String` type recognized as a *wildcard* type allows to have an equivalence on types for return and parameters (`R2`,`P3`).



(a) Required Interface (b) Candidate Service (c) Candidate Service

Fig. 4. Weather System

After the automatic Interface Matching for `TemperatureIF` and `Converter`, the syntactic distance between `TemperatureIF` and both candidate services is calculated, as shown in Table 5, being 1 for `TempConvert` and 0,5 for `Converter`. Thus, the suggested candidate for the next step of Behavioral Compatibility is the `Converter` service.

Table 5. Interface Compatibility Summary for `TemperatureIF` and the candidates

Operations of TemperatureIF	TempConvert Best Value*	Converter Best Value*
doFarenheitCentigrado	8	6
doCentigradoFarenheit	8	6
Total	16	12
Syntactic Distance	1	0,5

*Total Best Value 8 (based on `TempConvert` size)

These case studies show how a developer may gain specific and valuable knowledge about an application's context by the support of the Assessment Scheme. For each likely equivalence case automatically identified, there is a clear rationale that is also reinforced by the characterization within the four levels of compatibility. In addition, different scenarios of compatibility upon low levels may be analyzed by setting up other correspondences with the semi-automatic assistance based on the second part of the scheme. In this way, a certain web service may be saved from being early discarded as a potential candidate, but also a concrete validation is given for any change on correspondences, which become very helpful for a developer to understand the required adaptation effort to achieve the service integration.

[2] *http://www.w3schools.com/webservices/tempconvert.asmx?WSDL*
[3] *http://www.elguille.info/Net/WebServices/CelsiusFahrenheit.asmx?WSDL*

5 Conclusions and Future Work

In this paper we have presented details of a Selection Method which allows evaluating a candidate web service for its likely integration into a SOC-based application under development. This method is part of a larger process for discovery and integration of services, and provides a practical Assessment Scheme for Interface Compatibility where a synthesis of design and programming heuristics have been added, both to improve possibilities to identify potential matchings, but also to help developers to gain knowledge on the application's context for a candidate service. The syntactic distance metric provides a measurable value to mathematically support the candidate selection. Additionally, such selection might consider other aspects like *Quality of Service* parameters – e.g., performance, security, and so on.

The whole process of discovery, selection and integration has a fully support to achieve efficiency and reliability. Our current work is focused on exploring Information Retrieval techniques to better analyzing concepts from interfaces, which has been initially applied on the EasySOC approach. Another concern implies the composition of candidate services to fulfill functionality, which is particularly useful when a single candidate service cannot provide the whole required functionality. We will expand the current procedures and models mainly based on business process descriptions and service orchestration [11], [12].

References

1. Erickson, J., Siau, K.: Web service, service-oriented computing, and service-oriented architecture: Separating hype from reality. Journal of BD Management, 19(3), 42-54 (2008)
2. Bichler, M., Lin, K.: Service-oriented computing. Computer, 39(3), 99-101 (2006)
3. Flores, A., Cechich, A., Zunino, A., Polo, M.: Testing-Based Selection Method for Integrability on Service-Oriented Applications. In: 5th IEEE ICSEA'10. pp. 373-379. (2010)
4. Crasso, M., Mateos, C., Zunino, A., Campo, M.: EasySOC: Making Web Service Outsourcing Easier. Information Sciences, Elsevier. (2010)
5. Flores, A., Polo, M.: Testing-based Process for Component Substitutability. Software Testing, Verification and Reliability, p. 33 (2010), [early view press]
6. Stuckenholz, A.: Component Evolution and Versioning State of the Art. ACM SIGSOFT Software Engineering Notes, 30(1), 7-20 (2005)
7. Canfora, G., Di Penta, M.: Testing Services and Service-Centric Systems: Challenges and Opportunities. IT Professional, 8(2), 10-17 (Mar/Apr 2006)
8. Kung-Kiu, L., Zheng, W.: Software Component Models. IEEE Transactions on Software Engineering, 33(10), 709-724 (2007)
9. Jaffar-Ur Rehman,M. et.al.: Testing Software Components for Integration: a Survey of Issues and Techniques. Software Test., Ver., Reliab., 17(2), 95-133 (2007)
10. Alexander, R., Blackburn, M.: Component Assessment Using Specification-Based Analysis and Testing. Tech. Rep. SPC-98095-CMC, Software Productivity Consortium, USA (1999).
11. C. Peltz, Web Services Orchestration and Choreography. IEEE Computer, 36(10), 46-52, (2003)
12. Weerawarana, S.; et al., Web Services Platform Architecture: SOAP, WSDL, WS-Policy, WS-Addressing, WS-BPEL, WSReliable Messaging, and More. Prentice Hall PTR, (2005)