# Introducing Service-oriented Coverage Testing

Cesare Bartolini, Antonia Bertolino, Eda Marchetti
Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo"
Consiglio Nazionale delle Ricerche
Via Moruzzi, 1
56124 Pisa, Italy
Email: {cesare.bartolini, antonia.bertolino, eda.marchetti}@isti.cnr.it

## Abstract

*Testing of service-oriented systems is challenged by loose coupling and high dynamism, which are the founding characteristics of this emerging paradigm. Traditional test approaches needs to be revised for such systems, and in particular white-box techniques cannot be applied because services only grant black-box access. We introduce here a novel methodology, called SOCT (service-oriented coverage testing), that adapts the notion of coverage testing to the service-oriented domain, by exploiting the very features of service technology. In SOCT, both the probes inserted into the instrumented service code, and the retrieval of coverage-related information are themselves implemented as pure service invocations. A TCov service is published that makes available these coverage-related services through a WSDL interface. This simple idea elegantly enables the usage of practical test adequacy criteria also to service-oriented applications. In this paper a realization of SOCT is exemplified for coverage testing of BPEL orchestrations, and is illustrated on the case study of the Virtual Scientific Bookstore.*

## 1 Introduction

The emergence of the service-oriented development paradigm calls for novel approaches to verification and validation. Indeed, since services are pervasive in modern society, their malfunctions could heavily impact our everyday life and wellness. Therefore, appropriate guarantees need to be provided that they will behave properly and according to their published specifications. In fact, analysis and testing of services are actively investigated (e.g., [17, 10, 11, 6, 15, 5]).

The decoupling between the stage at which a service is developed and the stage at which it will later on be used by different stakeholders, and possibly integrated with other services, subverts the very idea of testing. Two are the peculiar aspects of service-oriented testing that descend from such separation: first, since a system is eventually obtained at run-time by black-box composition, it is difficult to predict in advance a representative sample of test invocations, unless a good crystal ball is at hand. A second peculiar aspect is that the developer and the user of a service are different people and unaware of each other, with any information exchange only happening via the published interfaces. At present, only the signature of services are generally made available, thus preventing on one side more thorough functional testing (which requires behavior specifications), and on the other the application of white-box coverage criteria (which need the source code).

To circumvent the above limitations, researches have focused mostly on the former need, and many proposals are being made to enrich the published interfaces of services with specifications of the dynamic aspects of the service invocation protocols. One quite natural way to extend a Web Service description is to use state machines, which can then be referred by Model-Based Testing approaches. As just one example of this research direction, we refer the Jambition tool [15] that uses a variant of state machines tailored for WSs.

To the best of our knowledge, instead, *no current proposal exists to address the second deficiency*, i.e., the unfeasibility of white-box approaches for services.

Essentially, coverage testing is realized by instrumenting the source code with additional function calls, called the *probes*, in correspondence of some program entities whose execution is monitored: for example, in branch coverage, a probe is inserted at the entrance of each branch. In traditional testing, coverage criteria have proved a practical and industrial-strength means to obtain testing thoroughness: program entities not

exercised by the executed test cases are highlighted, and additional test cases to target them can be sought.

It is unfortunate that, due to lack of access to a service internal implementation, coverage-based adequacy criteria cannot be applied. On the other hand, the separation between interface and code is at the basis of the flexibility and dynamic evolution features proper of this technology. Therefore, we certainly do not want to propose here to make available to the external world the source code of services for the very purpose of enabling white box testing.

In this work, we introduce a novel method to reconcile the flexibility, dynamism and loose coupling of SOA with the pragmatism of coverage testing. We propose to exploit the very feature of service-oriented technology also to implement coverage testing, realizing what we call *Service-oriented Coverage Testing* (SOCT).

The key idea at the basis of the approach is that we decouple the instrumentation phase (which is left to the service developer) from test execution and coverage measurement (which is left to the service integrator or provider). This is achieved by realizing both the probes inserted into the instrumented service code, and the retrieving of coverage-related information as pure service invocations. A `TCov` web service is published that makes available these operations through a WSDL interface. This simple idea elegantly enables the usage of practical test adequacy criteria also to service-oriented applications.

In the next section the basic notion of SOCT is described. The idea of SOCT can be realized in many different ways. In Section 3 we briefly describe how this could be realized for measuring BPEL coverage. The VSB case study described in Section 4 is used for some preliminary experimentation of the approach, summed up in Section 5. Related work is overviewed in Section 6, while some preliminary observations and future work conclude the paper in Section 7.

## 2 Service-oriented coverage testing

*If you can't beat them, join them*: service-oriented technology hinders the application of practical traditional testing techniques, such as white-box approaches [6]. However, what appears at first sight as a limitation of the technology can be switched into a further opportunity that it offers. The very features of service-oriented technology could be in fact exploited to deliver, at the tester's request, any desired coverage testing measure, in the form itself of a service.

We introduce here the novel methodology of *service-oriented coverage testing* (SOCT) for service testing. The idea behind SOCT, as illustrated in Figure 1, is simple and elegant. We assume first that a test service, `TCov`, is published, that provides at its interface the customary logging and reporting operations that are employed to realize coverage testing.

Service developers instrument services, similarly to the instrumentation stage of traditional coverage testing, by inserting probes in the points to be monitored. Differently from traditional coverage testing, however, in SOCT the probes themselves consist of invocations to `TCov`. These invocations consist of service request to log the entities that have been just covered.

Later on, service users and integrators who use the instrumented services and would like to apply coverage testing can launch a testing session. When the test session is completed, they can obtain the execution reports for the monitored entities by invoking `TCov`.

The `TCov` service supports the logging and the retrieval of the coverage-related data in a neutral way, agnostic of the specific adequacy criterion or test set-up. For example, it logs on-demand which entities are covered, but cannot deduce a coverage measure because it does not know the whole set of entities to be covered. Hence we assume that the Service User or Integrator uses a Test Coverage Analyzer to elaborate the data retrieved from `TCov` and infer a meaningful coverage measure.

The `TCov` service must be publicly accessible by both those who instrument the services and those who test them. Figure 2 provides the port type of a possible WSDL interface for a simple implementation of `TCov`. This (minimal) specification provides the following functions:

- *start* creates a new logging session, ensuring that there is no mix of existing and incoming data;
- *stop* ends the current logging session, for example it closes the database connection if necessary;
- *postAct* logs an execution unit which has been successfully executed;
- *faultAct* logs a failure of an execution unit;
- *coverageList* displays all logged data, without repetitions and in a non-chronological order;
- *coverageHistory* extensively displays all logged data, in chronological order and specifying the execution time.

Two issues must be highlighted with respect to the testing service. First off, we do not provide here a specification of the data to be sent to the service. The required data depend on a number of factors, including: The language and platform used for the service to

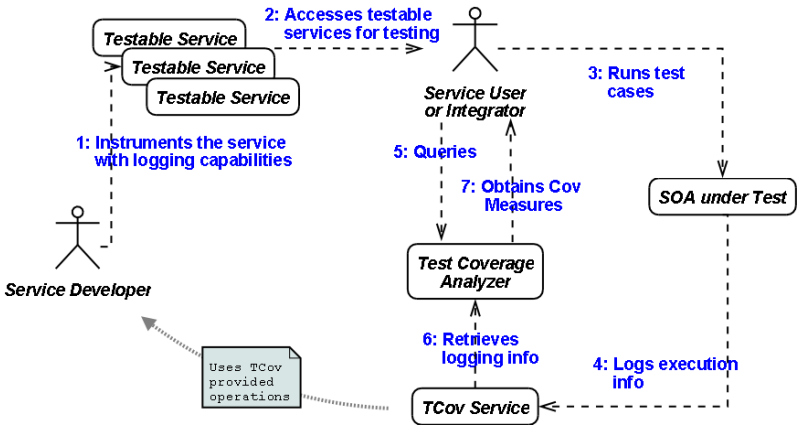


**Figure 1. The SOCT framework**

be tested, the technology behind the implementation of the testing service, the desired coverage and the requested level of detail. This technology dependency is not different from traditional coverage testing. Secondly, the `TCov` operations are divided into four logging operations and two query operations. The logging operations only have an input message and no output; this means that the service under test does not need to wait for an answer when sending logging messages, so the impact on execution performance by instrumentation is kept minimal.

The advantages of the proposed SOCT framework are evident: the testability of a service-oriented application is greatly improved, while loose coupling and late binding typical of the technology are kept untouched. Besides, decoupling between instrumentation (performed by service developer before deployment) and testing (carried on subsequently by service integrator or service provider) allows for maintaining the validity of coverage testing even in case of services evolution.

The methodology does not come without drawbacks. One major cost which could act as an hindering block is the extra effort for instrumentation required on the service developer's side. In the long term, cautious users could prefer testable services to completely closed ones, and this could convince developers to adopt the methodology. In the end, as for any other aspect of the open world philosophy, it is a matter of reaching a "social agreement" among all involved actors, in our case the service developers and the service users.

SOCT methodology is very general and paves the way for a variety of realizations. Indeed, in traditional programs, coverage testing has been applied to several different entities of control-flow or data-flow, has been performed at intra-procedural and inter-procedural levels, and has been specialized for the existing programming languages. The same types of variations could be identified for service-oriented applications.

In this paper, we provide a demonstration of the SOCT methodology in the orchestration of services (i.e., this would correspond to coverage testing at the inter-procedural level) for the widely used BPEL language and in particular we measure coverage of the basic message exchange operations (see Section 5 for details).

## 3 SOCT for BPEL

WS-BPEL [14], or simply BPEL, is a well-known OASIS standard for web service orchestration. It is composed of a set of XML languages (BPEL, variable property, partner link...) which together allow for using an algorithmic structure with web services; it also requires several platform-specific languages for graphical visualization (unspecified in the standard) and deployment. Additionally, the standard is extensible with additional XML languages (e.g., XPath) for extra functionalities, while other languages still (SOAP, WSDL, XSD...) can be needed for interaction.

The implementation of our SOCT methodology for BPEL requires the sending of specialized messages to the testing service `TCov`. In its simplest form, and without making any changes to the BPEL standard or the engine, this means adding a number of *invoke* activities; the number and positioning of these *invoke* activities, and the information contained therein, is strictly dependent on the entities monitored and on the desired coverage criterion. So, for example, definition coverage would require the insertion of an extra *invoke* at every *assign* or *fromPart* element, at least.

Before the logging *invoke* activity, the data to be

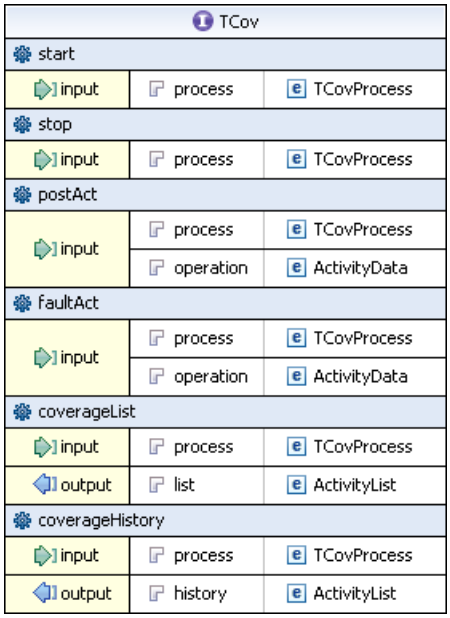


**Figure 2. A possible WSDL interface for `TCov`**

sent must be prepared, requiring an additional *assign* activity for each logging operation. So, to sum up, the simplest structure for applying the SOCT test paradigm to BPEL requires two BPEL activities per log unit, as well as the introduction of appropriate variables to store the data to be logged and a partner link to connect the BPEL process to the testing service.

The drawback of this approach is that the testing service (or at least a fake service with the same interface) *must* exist and be accessible from the BPEL process (even if coverage testing is not pursued), or else the whole process will fail. The workaround to overcome this problem would be to use logging activities conditionally, based on the value of some parameter; however, this would require an additional *if* activity for each logging unit. To avoid (visually) cramping the process with the logging facilities, we decided against this solution; however, doing so is a simple way of using a single BPEL process both when the testing service is required and when it isn't, without the need to have two different versions of the process.

In the current research, the design tool used for BPEL development is ActiveBPEL Designer version 4.1 and its related open source execution engine by Active Endpoints, Inc. [2].

## 4 Case study

To illustrate an example of the SOCT paradigm application to BPEL, we use a BPEL orchestration case study, which consists of a WS composition implementing a *Virtual Scientific Bookstore (VSB)*. VSB is a composite service which offers various functionalities related to scientific publications, including search of the ACM repository and visualization of the articles.

VSB has been realized from the composition of the three WSs described below: Pico, Google Citations, and ACM Search.

### 4.1 The Pico service

Pico is a web-based PHP application for scientific publication management currently in use at Scuola Superiore Sant'Anna [1]. Registered users can add, edit and remove the publications in the database, however its information is publicly available, therefore search results can be viewed by non-registered users. Beyond direct connection to the database, searches can be executed through the provided web service using SOAP invocations with RPC binding [18]. Of the many operations in Pico, only three have been used in this work:

**searchByAuthor** performs a search of all the publications in the Pico database including a given name in their list of authors.

- Inputs: authors' names (type: string); research sector. If not specified, then all research sectors will be included in the search (type: string); year (type: string); publication type (journal, conference proceedings...). If not specified, then all publications from these authors will be returned (type: string).
- Outputs: a list of references. If no publications relative to the requested authors were found an empty element is returned (type: sequence of record elements, from a custom XML Schema Definition).

**searchByTitle** performs the search according to the requested title.

- Inputs: publication title, or part thereof (type: string).
- Outputs: a list of references identical to the previous function.

**pdfGetter** recovers the file containing the printable form of the publication and returns its URL. It is important to note that this function does not actually return a PDF file, but a publicly accessible URL, so the requestor has a greater flexibility in choosing how to recover the actual file.

- Inputs: the name of the file (type: string).
- Outputs: the full URL where the file can be retrieved, or an empty value if it is not available (type: string).

### 4.2 The Google Citation service

Google Scholar (http://scholar.google.com) is a freely-accessible website useful for publication search. One of its most interesting features is that it displays the (supposed) number of citations that a given publication has received (according to the Google database).

As Scholar does not provide any web service support, we have implemented a web service performing a search on Google Scholar and receiving a web page in return. It contains a single function:

**citations** retrieves a web page from Google Scholar and searches for the number of citations of the given publication.

- Inputs: publication title (type: string).
- Outputs: number of citations, 0 if there were no results from Google Scholar, or no results correctly matching the title were found (type: integer); URL of the page where the information was retrieved (type:string).

### 4.3 The ACM Search service

Since the ACM scientific library does not provide any web service support, we have implemented a web service offering such a functionality. This service is composed of a single function:

**pdfGetter** queries the ACM repository with a search and receives a web page, which is then scanned to check whether the publication corresponding to the requested title is available on the repository. If so, this function returns a direct link to the PDF file, which can be used by the requestor. Note that this operation is not directly related to the operation of the Pico service with the same name; they are simply two operations which perform a similar function, but they are related to different WSDLs.

- Inputs: the title of the publication to search (type: string); login credentials (type: sequence of two strings, username and password).
- Outputs: the URL containing a direct link to the PDF file. It is a null value if no results were returned by the ACM search, or if none of the results matches the requested title (type: string).

### 4.4 The service composition: VSB

The final VSB service provides a set of functionalities to easily access the aforementioned WSs. Its execution is divided into two distinct steps. In the first step, upon a user request, a search in the Pico database is executed and the list of publications corresponding to the search criteria is presented to the user; then, the user must select a publication reference and an operation to perform on it, and the service will execute the requested operation and return the resulting value to the user.

Initially, the service composition expects two possible inputs: an author's last name and/or a publication title. If none is provided, then an exception is raised. If either one is provided, then the composition will accordingly invoke the Pico *searchByAuthor* or *searchByTitle* function, and will return the resulting reference list. If both inputs are provided, then the composition will call both functions in parallel, and the result will be the merging of the reference lists returned by *searchByAuthor* and *searchByTitle*. The merging is obtained by means of an XSL stylesheet, which can be used thanks to a function (*doXslTransform*) provided by WS-BPEL as an XPath extension. The resulting reference list is then shown to the user in an easily-readable form.

In the second phase of the service composition, the user must select one reference from the reference list. The function provided by the service composition for this purpose is called *selection*. Last, the user requests one of two operations, namely *citations* and *getPdf*. The requested operation is then invoked. The former invokes the Google Citations service, allowing the user to know the number of citations. The latter first determines whether the document is contained in the Pico repository (information available in the reference). If so, the *pdfGetter* function of the Pico service is invoked; otherwise, the ACM Search service must be called. However, since the ACM repository requires authentication, the user's credentials are sent along with the request.

However, in the current implementation of the case study, credentials are not actually used (in a real working environment, their absence should throw a fault, but fault management is covered in another part of the example).

After the request and the credentials are sent to the ACM service, the result will be the same as the invocation of the Pico *pdfGetter* function, that is a direct link to the document. Therefore, the composition will retrieve the document and present it to the user as the final output.

## 5 Experimental setup and results

Our case study focuses on a specific coverage criterion that we call *operation coverage*. By operations, we refer to all the BPEL activities which involve message exchanges, both with the human user and with other services. Technically speaking, we cover the activities involving *partner links*, namely: *invoke*, *receive*, *reply* and *pick*.

As said in Section 2, the scenario we envisage is that a service integrator wants to apply (operation) coverage testing on a BPEL process at the interprocedural level. Even though the BPEL code is owned, we are assuming for the purpose of demonstrating SOCT that the code of the orchestrated services is not accessible.

Note that in the following we play both roles (of the developer and the integrator), but in principle these are two separate entities, working independently of each other.

### 5.1 Operation coverage using TCov

We achieve the coverage using the methodology described in Section 3. This means that for each activity to be logged, two logging activities must be introduced: The first one to prepare the data for the logging service, and the second to send the data. Here we are playing the role of the service developer.

So, for example, after an invocation to the *searchByAuthor* operation, we introduce an *assign* activity which stores some logging information (name of the activity, name of the partner link involved and name of the WSDL operation, as well as the time) to a variable. Then, an *invoke* activity will pack the variable in a message and send it to the TCov service. Figure 3 shows the activities required to log an invocation to the *citations* operation.

When TCov receives the message, it stores the information in an internal database for later query.

Playing now the role of the service integrator, we have only knowledge of the BPEL process and of TCov WSDL interface. We invoke TCov to obtain the coverage information, possibly using an external module (referred to in Figure 1 as the Test Coverage Analyzer) to compute specific metrics.

```
<assign>
  <copy>
    <from><literal>invoke</literal></from>
    <to variable="operation">
      <query>activity</query>
    </to>
  </copy>
  <copy>
    <from><literal>google</literal></from>
    <to variable="operation">
      <query>operation/partnerLink</query>
    </to>
  </copy>
  <copy>
    <from><literal>citations</literal></from>
    <to variable="operation">
      <query>operation/operation</query>
    </to>
  </copy>
</assign>
<invoke operation="postOp" partnerLink="tcov"
        portType="tcov:tCov">
  <toParts>
    <toPart fromVariable="processData"
            part="process"/>
    <toPart fromVariable="operation"
            part="operation"/>
  </toParts>
</invoke>
```

**Figure 3.** *assign* **and** *invoke* **for logging a sample activity.**

### 5.2 Results

To test the operation coverage of the Virtual Service Bookstore process, we ran a test suite of 20 test cases. Each test case consisted of three steps:

1. a search request, using either authors' names, or keywords in the title, or both;
2. the selection of a specific publication among the ones returned by the search;
3. the request of either the number of citations the selected paper received, or the full PDF file.

For such a basic experiment, we hand-made the test cases, without using any automated test driver. Tests were run with the aid of an ad-hoc web interface. No failures emerged during the execution of this test suite.

Analyzing the TCov logs, we discovered that, over 9 total possible operations in the WSDL files of the various web services, 8 had been covered by our test suite, giving an 88% operation coverage. While the intent would have aimed at a 100% coverage, a deeper analysis revealed that the PDF files of all searched publications were present in the Pico database. This means that the ACM service has not yet been queried for PDF files by any of the executed test cases.

As in the philosophy of coverage testing, we used this feedback as a warning that our test suite was neglecting some functionality, and we then worked at enriching our test suite with additional test cases built ad hoc so to force the uncovered operation. We eventually reached 100% operation coverage. However, the new test unearthed an issue which prevented the PDF document to be correctly retrieved from the ACM repository. This was due to an imperfection in the request

to the ACM search service that had been overlooked.

## 6 Related work

With the wide spreading of web technologies, the field of web service testing has become a relevant and productive topic of research. Many proposals are available in literature, focusing on Verification and Validation (V&V) aspects, test case generation and execution, or test framework specification and assessment.

V&V approaches are usually applied to a service or to a service composition. Their purpose is to show that a service or a service composition conforms to a user-provided specification, or to evaluate the V&V methodologies according to their fault detection ability, or to assess some specific temporal properties. Recent proposals in this direction are [17, 16, 5] which are mainly focused on the conformance of a single service, and [19, 11, 7] which are specific to service composition and usually rely on the BPEL [14].

In particular, the `TCov` service can be efficiently integrated with methodologies for test case generation. In literature, different proposals provide strategies for test case generation, such as [9] which relies on Genetic Algorithms and [13, 3, 12] which focus on data flow testing criteria to test WS-BPEL applications. The test suite provided by these testing approaches can be executed, and coverage parameters such as the percentage of operations, branch, and data coverage can be evaluated within SOCT. This could be useful for improving the test suite effectiveness.

Testing web services also involves the definition and realization of a testing framework or specific testbed for early evaluation of quality aspects [8, 4]. These are usually used for deriving or validating the contracts of composite services, without accessing the actual invoked services. The coverage analysis provided by the SOCT methodology could be useful for analysis of the framework or testbed usage so as to improve service testing in a simulated environment.

## 7 Conclusions and future work

In this paper we presented a novel methodology for service oriented coverage testing, called SOCT. The main idea of SOCT is that the facilities for test coverage logging and measurement are directly implemented as service invocations. SOCT proposes itself as an approach complementary to the most widespread techniques of functional testing, and provides a methodology to revive white-box coverage criteria in a service-oriented ecosystem.

In this initial phase of the SOCT research, we focused on the `TCov` service, which allows WS developers to log data for specific purposes, for different coverage metrics, and at different levels of detail. For the moment, `TCov` is in a preliminary stage, and what we show here is a very basic WSDL interface. The structure of the service, the operations it offers, the messages exchanged, and the underlying technology will be refined or subject to change as the theory develops.

This approach offers several advantages. The first and foremost is the availability of an automated facility to enable the application of test adequacy criteria to the world of web services and web service compositions. The second benefit is that this result is achieved without the need to modify the execution engine, but only through modifications to the client. Generally, instrumentation requires some changes to the system running the application to be logged, so that operations are caught by the instrumentation before or after being actually processed; this is not necessary using the `TCov` service, because instrumentation is done on the side of the application to be tested.

Third, this technique is actually portable within BPEL, and platform-independent outside of it. So, for example, a BPEL process developed under the ActiveBPEL environment and annotated with `TCov` invocations will be entitled to use those same invocations if moved to the IBM Websphere system. More generally, any language or technology capable of using SOAP invocations can potentially be instrumented to use `TCov`.

For the sake of completeness, it should be pointed out that some of the available engines already provide facilities for logging various levels of detail without requiring the user to modify their core. ActiveBPEL, for instance, provides three different debug levels, namely "Full", "Execution" and "None". However, it is impossible for the user to gauge the logs to a desired detail or coverage level. `TCov` instead allows personalization of data collection, so that it can be easily adapted to different testing criteria such as operation coverage, branch or path coverage, data flow coverage and so on.

The main drawback of the approach is that, at the current implementation stage, each activity to be logged requires the introduction in the process of two additional activities, and this can lead to an overbloated process should a lot of logging be required. A secondary problem is that the service must be available even when logging is not required, unless two separate versions of the process are shipped, one with the logging activities and one without.

Both issues can be tackled with a solution which is planned as future work: The specification of an extension to the WS-BPEL standard, on the side of both the

web service developer and the execution engine. This extension should allow to use a single XML tag to annotate activities to be logged, and the engine (which must of course be improved to support the extension) will take care of sending the message to the `TCov` service. A mechanism should also be provided to globally enable or disable the logging facilities; this would allow to use the same BPEL process both in development and in deployment.

The above are technical considerations about the feasibility of the SOCT methodology. There are also social considerations involved, which are equally important for the successful realization of it. In the broad context, in fact, the vision depicted by the SOCT methodology is that the various stakeholders playing differing roles in the service-oriented domain will all agree on the usefulness and importance of making service applications more testable. As a consequence, they might want to abide by the procedures and conventions that monitoring service coverage by SOCT implies.

We are aware that requiring service developers to instrument the services for testing purposes is a burden that not everyone might be willing to assume. We are as well aware that reaching an agreement on the instrumentation and logging notations and procedures may be easier in some context, e.g. for orchestration, but might be more difficult in less centralized cooperations, e.g. for choreographies. We believe on the other hand that the extra work required of the developers is not overwhelming. Besides, concerning reaching a common specification for SOCT, this is in line with the trend in the open specification world, and we believe that in the way prospected here SOCT could be a *win-win* agreement.

## Acknowledgements

This work has been supported by the European STREP Project IST-26955 PLASTIC.

## References

[1] Pico 5. http://pico.sssup.it/, 2006.

[2] Active Endpoints, Inc. ActiveBPEL. http://www.activevos.com/bpel.php.

[3] C. Bartolini, A. Bertolino, E. Marchetti, and I. Parissis. *Data Flow-based Validation of Web Services Compositions: Perspectives and Examples*, volume 5 of *Architecting Dependable Systems*. Springer-Verlag GmbH, 2008. (to appear).

[4] A. Bertolino, G. De Angelis, L. Frantzen, and A. Polini. Model-based generation of testbeds for web services. In K. Suzuki et al., editor, *TestCom 2008*, volume 5047 of *LNCS*, pages 266–282. Springer, 2008.

[5] A. Bertolino, L. Frantzen, A. Polini, and J. Tretmans. Audition of web services for testing conformance to open specified protocols. In R. Reussner et al., editor, *Architecting Systems with Trustworthy Components*, number 3938 in LNCS, pages 1–25. Springer-Verlag, 2006.

[6] G. Canfora and M. Di Penta. Testing services and service-centric systems: challenges and opportunities. *IEEE IT Professionnal*, 8(2):10–17, 2006.

[7] H. Cao, S. Ying, and D. Du. Towards model-based verification of BPEL with model checking. In $6^{th}$ *Int. Conference on Computer and Information Technology (CIT 2006)*, pages 190–194, 2006.

[8] G. Denaro, A. Polini, and W. Emmerich. Early performance testing of distributed software applications. *SIGSOFT Softw. Eng. Notes*, 29(1):94–103, 2004.

[9] M. Di Penta, G. Canfora, G. Esposito, V. Mazza, and M. Bruno. Search-based testing of service level agreements. In *GECCO '07: Proceedings of the 9th annual conference on Genetic and evolutionary computation*, pages 1090–1097, New York, NY, USA, 2007. ACM.

[10] H. Foster, S. Uchitel, J. Magee, and J. Kramer. Model-based verification of web service compositions. In *18th Int. Conference on Automated Software Engineering (ASE 2003)*, pages 152–163. IEEE CS, 2003.

[11] J. García-Fanjul, J. Tuya, and C. de la Riva. Generating test cases specifications for BPEL compositions of web services using SPIN. In *Int. Workshop on Web Services Modeling and Testing (WS-MaTe2006)*, 2006.

[12] H. Lu, W. Chan, and T. Tse. Testing context-aware middleware-centric programs: a data flow approach and an RFID-based experimentation. *Proceedings of the 14th ACM SIGSOFT Int. Symposium on Foundations of Software Engineering*, pages 242–252, 2006.

[13] L. Mei, W. Chan, and T. Tse. Data Flow Testing of Service-Oriented Workflow Applications. In *TestCom 2008*, volume 5047 of *LNCS*, pages 371–380. Springer, 2008.

[14] OASIS WSBPEL Technical Committee. Web services business process execution language version 2.0. http://docs.oasis-open.org/wsbpel/2.0/wsbpel-v2.0.pdf, 2007.

[15] PLASTIC Validation Framework Tools homepage. http://plastic.isti.cnr.it/wiki/doku.php/tools.

[16] R. Siblini and N. Mansour. Testing web services. In *ACS/IEEE Int. Conference on Computer Systems and Applications*, 2005.

[17] W. T. Tsai, R. Paul, W. Song, and Z. Cao. Coyote: an XML-based framework for web services testing. In *7th IEEE Int. Symp. High Assurance Systems Eng. (HASE 2002)*, 2002.

[18] World Wide Web Consortium. SOAP version 1.2. http://www.w3.org/TR/soap/, 2007.

[19] J. Yan, Z. Li, Y. Yuan, W. Sun, and J. Zhang. BPEL4WS unit testing: Test case generation using a concurrent path analysis approach. In *17th Int. Symposium on Software Reliability Engineering (ISSRE 2006)*, pages 75–84, 2006.