Improving Trust in Composite eServices Via Run-Time Participants Testing[⋆]

Flavio Corradini, Francesco De Angelis, Andrea Polini[⋆⋆], and Alberto Polzonetti

Dipartimento di Matematica ed Informatica
University of Camerino
Via Madonna delle Carceri 9, 62032 Camerino - Italy
`firstname.lastname@unicam.it`

Abstract. The Service Oriented Architecture paradigm promises to open and integrate Public Administration offices in order to provide high-value e-services to citizens. Nevertheless to foster real usage of e-services by citizens, in majority still not fully acquainted with Internet technologies, it is necessary to put in place mechanisms to reduce as much as possible perceived system misbehavior. e-Services often handle personal and sensible data, therefore trust on the behavior of the system becomes of primary importance. In this paper, focusing on run-time composition of e-services, we provide an approach that reduces the possibility that the system will fail as consequence of interoperability issues among run-time discovered services, and after that sensible data have been provided by the citizen. The approach uses run-time testing to assess interoperability between services, and model-checking based techniques to reduce the number of test-cases to be applied. An exemplificative case-study is also illustrated and discussed.

1 Introduction

Interoperability and cooperation among Public Administrations (PAs) are nowadays fundamental aspects to promote the governance. e-Government [1] plays a basic role for a better delivery of government services to citizens, business and organizations, and for a more efficient management of the governance. For this reason public administrations promote e-services to automate their activities, to improve their cooperation and to provide faster and more efficient access to services. In this context the Service Oriented Architecture (SOA) is tailored for cooperation among different PAs covering different roles in administrative procedures. Practical benefits of SOA are today increasingly recognized permitting to improve business agility and to provide high quality services through the use of web service technology.

[⋆] The authors would like to thank Halley Informatica S.r.l. for its financial support.
[⋆⋆] Andrea Polini's work is partially supported by European Project TAS^3 (EU FP7 CP n. 216287) - `http://www.tas3.eu`

A web service (WS) can provide a real implementation of the e-Service concept. Technically a web service [2] is a piece of software available over the Internet that uses a standardized XML messaging system to interact with other WSs and general clients. Web services technologies solve low level interoperability issues through the use of open and standard XML-based formats (e.g. WSDL [6]) and protocols (e.g. SOAP [6]). Over such low level standards other application level standards have been defined allowing to describe the integration and cooperation of several services in order to fulfill a particular task. This is the case of standards for defining service orchestrations (e.g. WS-BPEL [4]), in which service compositions are defined assuming the availability of a central coordinator, or choreographies (e.g. [14]), in which such an assumption is not taken and services are integrated in a fully peer-to-peer fashion.

Application level standards seems particularly appealing for adoption within the e-government domain. In many cases full provisioning of e-government services could require and be represented through the integration and cooperation of services externally exposed by different PAs. In a near future a change of residence could be requested to the PA accessing to an orchestrated e-service which interacts with services provided by the different municipalities involved in the process. This would permit to a citizen to change her/his residence without the necessity of physically going to the various PAs offices.

Certainly in the described scenario, and in most scenarios involving PAs, many issues related to security, authentication, authorization, confidentiality and privacy of the process, and of the handled data, cannot be ignored. Indeed many standards are starting to emerge trying to provide an answer to such important issues (e.g. WS-Security [13]). Nevertheless the situation become even worse if we admit that services can discover each other at run-time, starting only then to interact. In such a setting many issues related to application level interoperability and integration emerge. Let's consider again the case of the change of residence. The services deployed by the two municipalities could be involved in the first mutual interaction of their existence. Wrong assumptions on data formats or application level protocols could lead to dangerous behavior and unpredictable results. In such a setting our work suggests the usage of run-time testing on discovered services in order to check service behaviour before real-usage. As consequence the interaction will start only if no mismatches are discovered and aborted otherwise. As better illustrated in the following of the paper the approach could also help to increase trust on service usage from citizens. The approach poses clear requests to run-time platforms and assumes the possibility of making testing invocations on running services. This behavior can certainly have dangerous consequences on running services that have to be take into account. Nevetheless these are quite general consequences of any run-time testing approach so they are not discussed any further here for reason of space.

In the next sections we detail the proposed testing approach and the hypothesis for its applicability. Next section discusses dynamic service composition and its influences on trustworthiness of services. Section 3 provide some technical background for the following material, and Section 4 details the various assumptions and phases of the approach. Successively in Section 5 we show how the approach can be applied on an exemplificative scenario. Finally we discuss in Section 6 some related works and we draw some conclusions and opportunity for future work in Section 7.

2 Dynamic Service Composition and Trustworthiness in the e-Government Domain

PA procedures to satisfy citizens needs can be quite complex. Full accomplishment of Citizen Directed Services (CDS), i.e. services that are explicitly defined to be directly used by the citizens, requires, in the general case, the execution and coordination of many different related tasks, often to be performed by various offices, possibly belonging to many different PA organizations. The involvement of different PA organizations has important consequences on the total time required to complete the provisioning of a single CDS. It is not difficult to identify PA services that last several days before being completed. Even worse there are cases, in particular in those countries in which PA organizations are rather loosely integrated, in which part of the coordination has to be directly carried on by the citizen, which has to visit different PA offices to collect, carry on, and return documents; in practice the citizen has to fix the inefficiencies in PAs integration and organization putting in place the needed coordination. This situation did not change much even with the advent and introduction of ICT within the PA sector. Indeed the integration of heterogeneous ICT infrastructures has been a really complex and general issue on the table for a long time, which, when feasible, often asks for extremely expensive solutions.

The Service Oriented Architecture (SOA) paradigm promises to revolutionize the world in ICT infrastructures integration. Thanks to the service abstraction concept and the adoption of open standards, different organizations can easily be integrated, starting to be interoperable without the necessity of sharing their internal business rules or data structures. Correspondingly a revolution can be easily foreseen in the provisioning of PA services to citizens. According to the new paradigm each PA will be abstracted as a set of provided services externally accessible, and implemented using one of the available technologies enabling the SOA vision; being Web Services (WS), and related technologies, certainly the most prominent choice at the moment.

Adopting the service oriented approach the real implementation of a CDS is derived through the integration and interaction of services externally exposed by each single involved PA. The integration and coordination of the different PA organizations is not anymore on the shoulders of the citizen. In this new scenario the citizen that needs a service can directly go to the offices of the specific PA responsible for providing the service, or can even simply connect to the PA related web site. No knowledge of “PA to PA” interactions is anymore required or will be evident to the citizen. The whole process is made real through the interaction of services exposed by the various PA organizations, with clear benefits also for what concerns total execution time.

CDS are typically implemented through the definition of so called orchestration. An orchestration describes, from the point of view of the orchestrator (director), how a set of participating services should be coordinated. Often the binding of a real service instance to a participant is defined at run-time depending on data provided to the orchestrator. For instance in a change of residence the town of provenance, and so the corresponding e-service, cannot be defined in advance.

In order to make run-time composition and interoperability easier, at least at the syntactic level, standard interfaces will emerge also in the PA sector. Definition of standard interfaces has also positive consequences on the market of services. Different developer can decide to provide implementation for defined services still having the possibility to interact at least at the syntactic level. On the other side run-time composition has important consequences on semantical interoperability increasing the risk of failure for a running orchestration.

Run-time misbehavior of service orchestrations can have subtle consequences in preventing real usage of e-government services. This is related to the important concept of user (citizen) perceived trust, which results strongly affected when the process handles sensible data in a perceived incorrect way. Particularly dangerous are those failures that manifest themselves after that sensible data have been provided. This is clearly the case for interoperability failures when run-time discovery and binding is permitted. Interoperability failure scenarios lead to frustrating sensations by the citizen that does not understand if her/his data have been modified or not. Even though a good infrastructure usually prevents from reaching inconsistent states, in which for instance the citizen looses the previous residence and does not acquire the new one, it is certainly probable that the citizen will not use the electronic service again.

Our proposal here, as described in detail in the next sections, is to introduce a short interaction trial for dynamically discovered services involved in a orchestration scenario, and before any request for sensible data to the citizen. During the trial a number of tests will be executed on the dynamically bound services in order to assess their behavior in the specific scenario. Therefore in case the trial ends with success the orchestration process will continue requesting to the citizen for relevant information. Instead if the trial ends in a negative way the orchestration ends, returning a message to the citizen saying that due to technical problems the service is currently not available. For sure the user will not be happy with this but its perceived trust will not be so much affected since no sensible data have been provided.

3 Technical Background

3.1 SOA, Web Services and Composition

Service oriented architecture is a computing paradigm and architectural style that leverage on the concept of service. In this vision services are software components that can be used through the net and composed to allow different applications to exchange data and participate in business processes.

The SOA paradigm can be implemented using different technologies among which the Web Service "technology suite" is certainly the most mature one. In particular in a WS setting services are described and published using the Web Service Description Language (WSDL [6]) and they exchange messages formatted according to the Service Object Access Protocol (SOAP) [6] specification. For service publication and discovery a registry is used. Typically in a WS setting the registry is an implementation of the Universal Discovery Description and Integration (UDDI [6]) registry specification.

Above such basic layer, focusing on individual description of service characteristics, various standards have been defined to describe service compositions. Among these standards the Web Service - Business Process Execution Language (WS-BPEL) is certainly the most mature and supported one.

Using WS-BPEL several services can be coordinated by a business process. This process controls service execution providing programming constructs such as sequence, parallelism, loops, conditional and case statements. A WS-BPEL orchestration introduces roles to be played by the services involved in the conversation. This can be done using *PartnerLink* definitions which show the WSDL ports provided by each service in order to exchange messages with other services. Messages are handled by three basic activities: *invoke*, *reply* and *receive.* These respectively correspond to the action of sending and receiving a message in a request-response scenario, the action of sending a response message, and the action of receiving a request message.

3.2 Model Checking

Model checking [7] is a very effective technique to deal with formal analysis and verification of complex software system specifications. A model checker is able, given an operational model of the system and a property that states specific requirements on the same model, to show if the property actually holds or not. Moreover in case the property is not satisfied the model checker points to a counterexample that shows a precise model execution that violates the property.

Since its first inception many tools have been proposed and developed, nevertheless all of them share the same principle. In particular the system is represented by an operational model defining a system state space and the transitions among such states. The property instead is generally specified through a logical formula such as a Linear Temporal Logic (LTL) formula ([8]). Given the model and the formula the model-checker exhaustively explores all the possible system successive configurations looking for possible violations of the formula. When a violation is detected the model checker reports to the user the sequence of decisions and actions that it took, during the exploration, to reach the falsifying configuration.

Model checking has been demonstrated to be a really powerful approach to verify system properties, nevertheless it generally suffers from the well known state-explosion problem, i.e. the number of possibly different system configurations can become too big to permit a complete state exploration. To overcome this problem many heuristics have been defined in some cases resulting in almost-precise approaches. The state-explosion problem is particularly relevant when the model is augmented, as it is for the case considered in this paper, with data ranging over wide domains such as for instance integer.

3.3 Genetic Algorithms

Genetic algorithms [11] are a technique that mimics natural-evolution processes to find approximate or exact solutions in search problems with large solution spaces, and that are not amenable to exhaustive search. The technique borrows concepts from evolutionary biology like inheritance, mutation, selection, and recombination to evolve candidate solutions, represented by a set of chromosomes, toward an acceptable solution for the problem.

A generic genetic algorithm requires the definition of a representation both of the solution in terms of chromosomes and of a function (fitness function) to be used to compare the evolving solutions. A fitness function is the objective to reach for a right solution. Therefore, a major issue with this kind of algorithms is the design of the structure of the solution and of the method for its evaluation using the fitness function as the objective. Other parameter that may have critical importance are the size of the population, the type of recombination, and the mutation rate.

Like other Artificial Intelligence techniques, these kind of algorithms can be applied to many software engineering fields. Their application as search strategy heuristics for model checking allows the exploration of large state spaces [10] starting from candidate solutions found by the algorithm. Application in software testing instead concerns data generation. In designing test cases is desirable to find test inputs and test oracles to verify correct behavior of programs. Find test data may be a time consuming task for developer or an hard task if we deal with large domains. Numerous attempts try to overcome these drawbacks automating the generation of these kind of data.

4 The Approach

Run-time discovery and binding is certainly a useful and required characteristic of PAs integration through orchestration definitions. Nevertheless its usage can drastically affect interoperability and countermeasures have to be considered in order to reduce the risk of run-time failures due to interoperability factors. The approach we propose suggests the usage of testing to reduce the risks that a failure is discovered during the processing of a client request, and after she/he has provided sensible data to the service.

Applying our approach in order to derive a test suite for possibly discovered services, the orchestration developer has to put in place various steps. Some of these steps can be quite complex and time consuming. Nevertheless they have to be executed once and for all before the final deployment of the orchestration. Such complexity is justified since it is important to reduce as much as possible the number of test cases to be executed at run-time. Certainly this contrasts with the idea that bigger test suites provide a more comprehensive verification. Indeed the effectiveness of the approach strongly depends from the definition of a good test case selection criteria focusing on interoperability issues. In the following subsection we discuss the different steps composing the approach thus to provide answers to the these issues.

4.1 Defining BPEL Orchestration for Run-Time Testing

The approach we propose has important consequences on the structure of the orchestration specifications, that the developer has to take into account. In particular in order to carry on a testing trial on run-time discovered services, before that sensible data are provided by the citizen, it is necessary to organize information requests, within the orchestration, in a two steps procedure. In the first step the orchestration only asks for information enabling the discovery and identification of the services to be involved in the interaction. Then all the discovered services will be submitted to a trial according to the defined test suites. In case no interoperability issues are highlighted the orchestration asks for sensible data to the citizen. Instead, in case some interoperability threats are identified, the orchestration replies to the citizen saying that the service is currently not available- at this point no sensible data was provided.

Considering for instance the change of residence example, this would mean that the orchestration has to be organized in order to firstly ask for the coming residence but not for name and other personal information. Having this information it is possible to identify the service that will participate in the orchestration i.e. the municipality to be contacted. Successively personal information will be asked in the second phase and only if the testing trial ends with a success.

In our research we have investigated many citizen directed services and for all of them it seems possible to split the information request in a two phase procedure according to our requirement. Nevertheless in case this would not be possible the orchestration could be organized in more than two steps asking in each step the less information as possible.

4.2 Model Checking BPEL Specifications

Our approach applies a counterexample-based technique that starting from a model and a property is able to derive a test case from a counterexample for that property. A counterexample provides a trace of events for which the property does not hold. Therefore, if we declare that the property $\neg P$ should hold for the system model, the technique generates, in case it exists, a trace for which P holds.

A BPEL flow is typically data-driven. This can be a big problem for a model-checking technique since it can easily lead to a state-explosion situation. To solve this problem some approaches remove data from the model and then choose non determinism to handle conditional choices within the orchestration (i.e. they randomly select a branch of a conditional statement). In our work we use a data generation technique to generate sets of data to drive the model checking phase. In this manner we simulate correct interactions between the orchestrator and the external services, given the selected data. To generate the needed sets of data we apply genetic algorithm approaches as detailed in Section 4.3.

Test-cases are derived from counter-example generated by the model-checker when reachability properties are specified. In this way we can highlight all the possible paths that can bring from the start of the BPEL process to any possible final state. The derived paths include the interaction actions among the orchestrator and the composed services. For such steps also data are defined according to those provided by the applied genetic algorithm.

In our work we use the model checker BOGOR [16] that is an extensible model checking framework designed to support general purpose and domain-specific (via customization) model checking.

Finally to translate a BPEL orchestration in a format accepted by BOGOR we use BPEL2BIR [5]. BPEL2BIR is a tool that can translate a BPEL specification into a BIR model in order to apply model checking techniques. We have extended this tool to allow the generation of data-driven counter-example suitable for our purposes.

4.3 Test Data Generation

Test data used to drive the state space exploration are generated using a biologically inspired tecnique implemented as a genetic algorithm. The fitness funtion is used to compare different test suites and is tailored on the concept of interaction adequacy criteria and interaction coverage. We define such concept exploiting classical definitions of branch testing, and path testing [15].

Interaction adequacy criteria. Let T be a test suite for a BPEL process P. T satisfies the interaction adequacy criterion for P iff, for each interaction I of P, there exists at least one test case in T that causes execution of I (Where a interaction represents a message exchange between the orchestrator and a composite service).

Interaction coverage. The interaction coverage $C_{Interaction}$ of T for P is the fraction of interaction of process P executed by at least one test case in T.

$$C_{Interaction} = \frac{\text{number of executed interactions}}{\text{number of interactions}}$$

T fully satisfies the interaction coverage adequacy criterion if $C_{Interaction} = 1$

A solution that covers the greatest number of interactions in the process has the highest fitness function value. Interactions are counted with their occurrences. So in case the BPEL Process presents loops it is necessary to define limit to the number of execution.

Our approach aims to merge the space exploration task with data generation features. This is possible if the model used for the state space generation is expressive enough to support an exploration driven by data. Indeed this is the case of BPEL processes.

Data generation terminates when generated data covers a specific amount of interaction in the model, also respecting eventually reachability properties expressed on the model iself. With respect to other approaches in this manner we can derive from the model checker traces input data (and oracle if the model fully specify participant's behaviour) observing interactions among the various participants.

4.4 Test Suite Generation

The combined use of model-checking and genetic algorithms techniques permit to derive a set of traces that are characterized by an high coverage of the interaction actions among the BPEL process and the composed services, as discussed in the previous sections.

Given a selected trace in order to derive test cases for the different services involved we project the trace, as in Figure 1, with its relative data, over the various participants. In this way we can isolate behavior snippets that can be used for test purpose. Successively we combine all this pieces in a sequence of invocations for a specific orchestration participant. The fact that the traces were derived and evaluated taking into account the number of enclosed interaction actions guarantees that selected test cases are certainly relevant for what concerns the detection of interoperability threats.

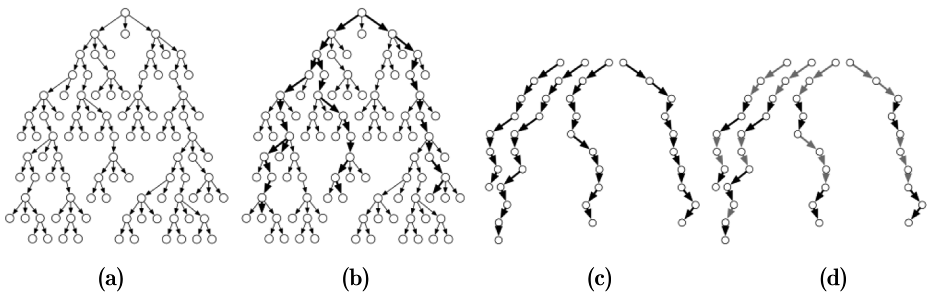


Fig. 1. (a) Unfolding of the system traces, (b) Example of traces selected by the property verification with specific data, (c) Some traces isolated from the tree, (d) Trace snippets for a participant (the gray arrows are interactions with the context)

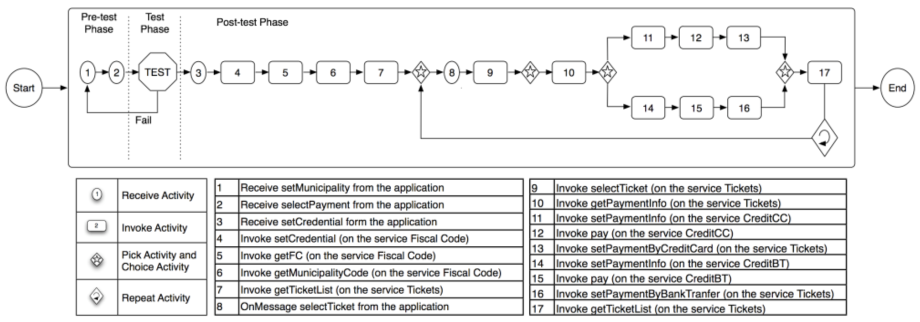
5 Exemplifying Scenario

In this section we introduce a simplified example in the e-government field to better explain our approach. We consider a composite service that represents the back-end of an on-line payment of fines for traffic violations.

The system allows the user to list traffic violations inficeted by a given municipality. The user is able to pay a fine on-line (interacting with a payment service) or off-line (via bank transfer). The modality chosen for the payment modifies the behavior of the process affecting the methods that must be invoked to notify the payment to the municipality. We want to test this service using an adequate test suite that is expressive enough to cover all possible interactions. For simplicity and to explain our approach we refer to a single test suite for a service, but the aim is to produce test suites for all the services involved in the composition. The system for managing traffic violations is composed by an orchestration of four different services:

- *Fiscal Code*: given user data returns the user Italian fiscal code (similar to US SSN) that is used as unique identifier for the user and the municipality code used to identify an Italian municipality
- *Tickets*: A service that returns the sanction (here called ticket) for traffic violation within a given municipality
- *CreditCC*: A service that allows the payment via credit card of a ticket
- *CreditBT*: A service that allows the payment via bank transfer

The four services are orchestrated by a BPEL process that handles the interaction between the user and the services as depicted in Figure 2.



①	Receive Activity
▭	Invoke Activity
◇	Pick Activity and Choice Activity
◇	Repeat Activity

1	Receive setMunicipality from the application
2	Receive selectPayment from the application
3	Receive setCredential form the application
4	Invoke setCredential (on the service Fiscal Code)
5	Invoke getFC (on the service Fiscal Code)
6	Invoke getMunicipalityCode (on the service Fiscal Code)
7	Invoke getTicketList (on the service Tickets)
8	OnMessage selectTicket from the application

9	Invoke selectTicket (on the service Tickets)
10	Invoke getPaymentInfo (on the service Tickets)
11	Invoke setPaymentInfo (on the service CreditCC)
12	Invoke pay (on the service CreditCC)
13	Invoke setPaymentByCreditCard (on the service Tickets)
14	Invoke setPaymentInfo (on the service CreditBT)
15	Invoke pay (on the service CreditBT)
16	Invoke setPaymentByBankTranfer (on the service Tickets)
17	Invoke getTicketList (on the service Tickets)

Fig. 2. Sample BPEL Process

The interface exposed by the process include the methods that external applications can invoke to perform payment of fines. A typical scenario foresees the invocation of the method *setCredential* that supply the process with the necessary information about the user, and the invocation of the method *setPayment* to select the modality of payment that will be used in the following interactions.

According to our approach the BPEL orchestration will ask, in a first step, for information concerning the municipality (so to identify the Ticket service) and the way of payment (so to identify the payment service). These information permit to the orchestrator to identify all the services that will interact at run time in order to fulfill the fine payment. After the identification the run-time discovered services will be submitted to a testing trial using the corresponding test suite derived applying the approach described in this paper.

So for instance in case the user specify town M and payment via Credit Card issued by bank B, the orchestration will retrieve a reference to service *Ticket* exposed by M and to service *CreditCC* exposed by B and will start the trial using the test suites defined for the two services. Obviously no trial will be conducted on services of type *CreditBT*. It is worth noting that no data directly related to the user have been provided so far.

In case during the trial phase no errors are highlighted, the process continues asking for personal information such as Fiscal Code. Personal data are then provided to the *Fiscal Code* service (which is a statically bound service so no test are executed on it). The returned fiscal code is passed to the *Ticket* service of the municipality M and a possible list of fines are reported. The used select the fine she/he wants to pay and credit card detail are then requested and passed to the *Credit* service provide by B.

On the other side if the trial discover a problem the process terminates saying that the service is temporarily unavailable. At this point failure details will be logged and reported to the technical teams for further investigation.

6 Related Works

Direct interferences of interoperability issues on service trustworthiness has been explicitely reported in [18]. Our approach combines various techniques in order to apply run-time testing for services to be composed in BPEL processes. To the best of our knowledge there are no directly related approaches in literature, and for which we could provide a comparison. Nevertheless we derived our idea from many different sources.

In particular derivation of test cases from counter-examples has been proposed by [3], [12]. A discussion on model-checking and genetic algorithms can be find in [10]. Finally use of Model-checking techniques for the analysis and verification of BPEL orchestrations has been proposed in [9], [17].

7 Conclusions and Future Work

Citizen trust is a major requirement for real take-off of e-government procedures. SOA promises to revolutionize the way in which e-services will be provided to citizens permitting to easily integrate different PA organizations in a dynamic and transparent way. Nevertheless dynamic discovery of services can increase the risk of failure of composite services with strong consequences on citizen trust. In this paper we propose an approach to reduce the risk or run-time failure due to interoperability issues and after that sensible data have been provided by the citizen. Doing so consequences of failures on citizen trust should be much less impacting. The approach we propose is based on derivation of test cases from BPEL specification applying Model-Checking and Genetic Algorithm techniques in order to have small but powerful test-suites.

The framework have been experimented with some exemplifying scenarios, providing comforting results. In the future we intend to continue the experimentation, also in order to compare the suggested test-case derivation techniques with other Model based approaches discussed in the literature. Also the real impact on consequences of citized trust need to be empirically evaluated.

References

1. AA.VV. The Role of eGovernment for Europe's Future. Technical report, Commission of the European Communities, Brussels (September 2003)
2. Alonso, G., Casati, F., Kuno, H., Machiraju, V.: Web Services–Concepts, Architectures and Applications. Springer, Heidelberg (2004)
3. Ammann, P., Black, P.E., Majurski, W.: Using model checking to generate tests from specifications. In: ICFEM 1998 (1998)
4. Andrews, T., Curbera, F., Dholakia, H., Goland, Y., Klein, J., Leymann, F.k., Liu, K., Roller, D., Smith, D., Thatte, S., Trickovic, I., Weerawarana, S.: Business Process Execution Language for Web Services Version 1.1 (July 2002)
5. Bianculli, D., Ghezzi, C., Spoletini, P.: A model checking approach to verify BPEL4WS workflows. In: SOCA 2007, pp. 13–20 (2007)
6. Curbera, F., Duftler, M., Khalaf, R., Nagy, W., Mukhi, N., Weerawarana, S.: Unraveling the web services web: an introduction to SOAP, WSDL, and UDDI. Internet Computing, IEEE 6(2), 86–93 (2002)
7. Clarke Jr., E.M., Grumberg, O., Peled, D.A.: Model checking. MIT Press, Cambridge (1999)
8. Emerson, A.E.: Temporal and modal logic. Handbook of theoretical computer science formal models and semantics, vol. B, pp. 995–1072 (1990)
9. Foster, H., Uchitel, S., Magee, J., Kramer, J.: Model-based verification of web service compositions. In: ASE 2003, pp. 152–161 (2003)
10. Godefroid, P., Khurshid, S.: Exploring very large state spaces using genetic algorithms. International Journal on Software Tools for Technology Transfer (STTT) 6(2), 117–127 (2004)
11. Holland, J.H.: Adaptation in natural and artificial systems. MIT Press, Cambridge (1992)
12. Huang, H., Tsai, W.T., Paul, R., Chen, Y.: Automated model checking and testing for composite web services. In: ISORC 2005, pp. 300–307 (2005)
13. Nadalin, A., Kaler, C., Hallam-Baker, P., Monzillo, R.: Web Services Security: SOAP Message Security (August 2003)
14. Peltz, C.: Web services orchestration and choreography. Computer 36(10), 46–52 (2003)
15. Pezze, M., Young, M.: Software Testing and Analysis: Process, Principles and Techniques. Wiley, Chichester (April 2007)
16. Robby, Dwyer, M.B., Hatcliff, J.: Bogor: an extensible and highly-modular software model checking framework. In: ESEC/FSE 2003, pp. 267–276. ACM, New York (2003)
17. van Breugel, F., Koshkina, M.: Models and verification of bpel. Technical report, York University (2006)
18. Zhang, J.: Trustworthy web services: Actions for now. IT Pro. 7(1), 32–36 (2005)