Towards self-diagnosing Web Services

Liliana Ardissono, Luca Console, Anna Goy,
Giovanna Petrone, Claudia Picardi, Marino Segnan
Dip. Informatica – Università di Torino
liliana,lconsole,goy,giovanna,marino,picardi@di.unito.it

Daniele Theseider Dupré
Dip. Informatica
Università del Piemonte Orientale
dtd@mfn.unimpn.it

***Abstract*— Distributed software systems would benefit from autonomous fault management capabilities, but current practice is only based on handling exceptions without attempts at identifying causes for them. This paper is a step toward Web Services with autonomous diagnostic capabilities. It provides a novel context of application for model-based diagnosis, a contex which motivates a partially distributed approach. We consider complex services, built as a composition of simpler ones, and we associate a diagnoser with each component service, and a global diagnoser with the complex one. We characterize local diagnosers, based on abstract models of individual services, and we present the coordination protocol adopted by the global diagnoser.**

I. Introduction

Service Oriented Architectures [9] and standard languages for the publication and invocation of Web Services, such as WSDL [13], enable the exploitation of heterogeneous software by abstracting from the features of the deployment environment of applications. On top of these basic communication languages, standard Web Service composition languages, such as BPEL [2], are being defined to support the development of complex applications based on the orchestration of simpler ones. Moreover, in the Semantic Web community (see, e.g., [5], [8]), languages and frameworks are being defined to support a suitable specification of services and intelligent service cooperation (e.g., see [7]). The growing worldwide acceptance of these standards is an excellent start for a realistic integration of services in the Web, as well as in Enterprise Application Integration, which represent two mainstreams of software development in the next future [1].

However, several issues have to be addressed in order to enable the effective integration of non trivial applications. In fact, rather straightforward solutions are currently adopted to support the reliability of services. The ability to detect and isolate faults during service execution and to apply recovery actions in an efficient and effective way would be very desirable, especially for the creation of complex services from simpler ones whose implementation is not publicly available.

In this paper we propose a framework for adding diagnostic capabilities to Web Services, using a model-based perspective [4]. The goal is to design **self-healing** services which guarantee autonomous diagnostic and recovery capabilities. When defining a complex service, composed of simpler ones, we propose to add to each service S a local diagnoser which relates hypotheses about incorrect outputs of S to a misbehaviour of S itself, or to incorrect inputs from other services. A global diagnostic service is then associated with the complex service. It coordinates the local diagnosers, exchanging messages with them and, without relying on any information about the internal structure of the sub-services, it can in turn compute diagnoses at the level of the global service. The same idea can be adopted recursively when the global service is used as a component of a more complex service. In the paper we discuss a protocol for a global diagnostic service, and we characterize the operations that local diagnosers must support in order to comply with such a protocol. The goal is the identification of the faulty service, not debugging the service itself. In addition, the local diagnoser may identify a part of the service which is claimed to be responsible for the fault.

We choose to adopt an approach based on the introduction of a global diagnostic service because this enables to recursively partition Web Services into aggregations of subservices, hiding the details of the aggregation to higher-level services. This is in accordance with the privacy principles which allow to design services at enterprise level (based on intra-company services) and then use such services in extranets (with other enterprises) and public internets. The global diagnostic service only needs to know the interfaces of local services and share a protocol with local diagnosers.

Section II sets the context of Web Service diagnosis; section III introduces the approach we adopted to model services; section IV introduces the protocol for the global diagnostic service, and characterizes local diagnosers;section V overviews existing research and future work on the topic.

II. The context: Web Services diagnosis

Currently, fault handling in Web Services (WSs for short) is not performed in a satisfactory way as it basically relies on the handling of exceptions raised by invoked services; no attempt is made to identify the causes of faults. This may be a limitation, especially in complex services, composed of several Web Services where problems might be caused by the interaction between services and where the absence of specialized diagnostic capabilities usually imposes the execution of coarse grained repair actions when errors occur.

We show our viewpoint on an example adapted from [12]. A bookshop offers a Web-based catalog whose user interface is implemented as a Web Service (Catalog WS) interacting with the main backoffice Web Service of the bookshop (Bookshop WS). When a customer selects a book, the Web Services exchange the following messages (see Figure 1):

- The Catalog WS sends an order of a book to the Bookshop WS (message 1).

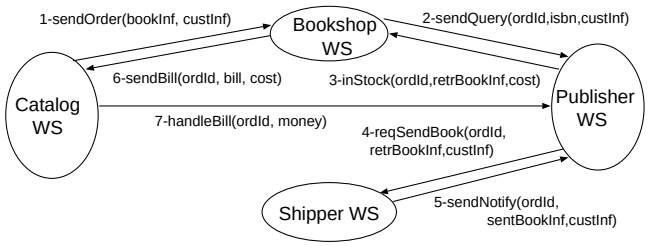

Fig. 1. Collaboration diagram for a book sales scenario.

- The Bookshop WS retrieves the ISBN number of the book. Then it sends a request to the Publisher WS to deliver a copy of the book to the customer (message 2).
- The Publisher WS retrieves book details from the ISBN number and notifies the Bookshop WS that the book is available (3). Then, the Publisher WS asks the Shipper WS to carry one copy of the book to the customer and gets back the delivery acknowledgment (messages 4 and 5). The physical delivery of the book is not shown in the figure because it is not an electronic operation.
- The Bookshop WS sends the bill to the Catalog WS (6).
- Finally, the customer pays through the Catalog WS that notifies the Publisher WS (7).

Now, suppose the customer receives the wrong book. Which service provider is responsible and should be charged with the extra delivery costs? The problem might be caused by errors occurring during the execution of different Web Services and the identification of the faulty one is not obvious, unless suitable diagnostic reasoning is employed.

In the paper we develop a framework for tackling such a problem. Although we consider complex services based on the cooperation of other services, we do not make assumptions on how the cooperation is orchestrated. We will see that the global diagnostic service needs not know in advance how the individual services interact. This means that the cooperation could be based on the adoption of a workflow, or that Semantic Web descriptions of the services and interaction protocols [8] may be exploited for intelligent composition (see Section V).

III. Architecture

We propose a partially distributed approach, where several **local diagnosers** $A_1, \ldots, A_n$ cooperate with a **global diagnostic service** ***D***. Each local diagnoser A_i is responsible for a Web Service W_i (or a set of Web Services), while ***D***, as we shall describe later, puts together information from local diagnosers and selects which local diagnosers to question further in order to diagnose problems.

Following the Model-Based Diagnosis paradigm, inferences of each local diagnoser A_i are based on a model M_i of the service(s) it is responsible for. Such a model is an abstraction of the computation carried on by the service.

In particular, as it is common in workflow modelling [12], such a computation is represented as a set of **activities** with **input** and **output** variables (in this paper we limit the discussion to stateless systems). The activities invoked on different service providers correspond to sending messages (e.g. WSDL messages [13]); variables contained in these messages will be called *interface variables* of the sender and receiver WSs.

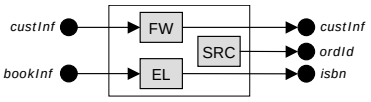

Fig. 2. Dependencies for an activity.

Input variables of activity a represent information used by a; output variables represent information propagated to another activity b, and which could have been produced by a. The model could be given in terms of three templates, *forward* (FW for short), *source* (SRC) and *elaboration* (EL), which distinguish whether an output variable is a copy of an input variable (i.e. the information is used by a and b, but not modified by a), or it is created by a, e.g. retrieved from an internal database, or it is computed by a depending on some of its input variables. Figure 2 illustrates, for the example in figure 1, the activity of the Bookshop WS which receives message 1 and sends message 2.

From the representation described above, a diagnostic model M_i is provided as follows for each Web Service W_i. Each activity is, at least for the local diagnoser,[1] a smallest diagnosable unit, i.e. it corresponds to a component in Model-Based Diagnosis [4]. For each WS variable v represented in the model (as input or output of an activity), a corresponding binary variable v' is introduced in the diagnostic model. The *ok* (resp., *ab*) value for v' represents the fact that in a given execution of the service, v has the expected value (resp. a different value with respect to the expected one). For each activity we consider an *ok* and a *fail* behaviour, and, for each of them, a relation constraining values of variables in the model under the assumption that the activity is ok or not.

If the model of an activity is given in terms of FW/SRC/EL blocks, its model can be derived from a default model of each template. For EL blocks (and SRC, which are ELs with no input) a default model $\mathsf{EL}_{\mathrm{dfl}}$ is the following:

- In the *ok* mode, if all inputs are *ok*, the output is *ok*. Otherwise, the output is unconstrained.
- In the *fail* mode, the behaviour is unconstrained.

FW blocks can be distinguished from EL blocks assuming, as a default model, that they cannot fail, i.e. that their behaviour in the *fail* mode is the same as in the *ok* mode.

More specific models can be provided. E.g., knowing that an EL activity computes an injective function would exclude, in the *ok* mode, the output to be *ok* if a single input is *ab*.

Each WS is endowed with a set of alarms that may be triggered depending on certain conditions. Each local diagnostic agent A_i is informed about the alarms in W_i and their triggering conditions, expressed in a way that can be related to the model. A typical triggering condition for an alarm a is a mismatch of two WS variables x and y (e.g. in the book sales scenario the Bookshop WS may check whether the information on the book provided by the Catalog WS matches the information on the book found by the Publisher WS).

[1] As we shall discuss later, the local diagnoser may want to hide the internal architecture of the corresponding WS.

In the corresponding part of the diagnostic model, a binary variable a' is introduced to represent whether the alarm is raised ($a' = ab$) or not; a' is related to x' and y' as follows:

- a' is *ok* if both x' and y' are *ok*.
- a' is *ab* if one of x' and y' is *ab*.
- a' can be *ok* or *ab* if both x' and y' are *ab*.

A Web Service W_i may have been designed with a set of alarms that make it diagnosable as much as possible. If this is not the case, in order to enhance diagnosability without modifying its implementation with additional alarms, the local diagnoser A_i records messages sent and received by W_i, and possibly those internal actions that correspond to messages (in case W_i is in turn a composition of services). A_i can be designed (possibly after diagnosability analysis on its model of W_i) to perform predefined checks on such messages; such checks will not be performed by W_i when it runs (W_i could therefore remain unchanged); they will be performed by A_i if and when it is awakened. These predefined checks will be called *checkpoints*. Similarly to alarms, a checkpoint c provides a binary piece of information c', that can be related to the WS model in the same way.

IV. The Diagnostic Protocol

We first give an informal description of the interaction between local diagnosers and the Diagnostic Service ***D*** (section IV-A). Then we formalize a protocol for ***D*** (section IV-B). As to local diagnosers, we characterize their operations, without providing specific algorithms (section IV-C).

A. Interaction among diagnosers

The global Diagnostic Service ***D*** does not initially have any information on the individual Web Services. Its main job is to put together information coming from local diagnosers and to select which local diagnosers to question further in order to obtain the desired result.

When an alarm is raised in a Web Service W_i, the local diagnoser A_i receives it. A_i must explain it, and provide ***D*** with the results. Each explanation may ascribe the malfunction to failed internal activities and/or abnormal inputs. It may also be endowed with predictions of additional output values, which can be exploited by ***D*** in order to validate the explanation by acquiring new observations that may falsify the hypothesis. When ***D*** receives the output of a local explanation from a local diagnoser A_i, it can proceed as follows:[2]

- If a Web Service W_j has been blamed of incorrect outputs, then ***D*** can ask its local diagnoser A_j to explain them. A_j can either reject the blame, explain it with an internal failure or blame it on someone else.
- If a fault hypothesis by A_i has provided additional predictions on output values sent to a Web Service W_k, then ***D*** can ask A_k to validate the hypothesis by checking whether the predicted symptoms have occurred, or by making further predictions.

[2]We assume that each interaction among Web Services is identified by a *conversation id* which is mentioned in each information exchange between local diagnosers and ***D***, in order to identify a diagnostic session.

Hypotheses are maintained and processed by diagnosers as *partial assignments* to interface variables and behaviour modes of the involved local models. Unassigned variables in partial assignments represent parts of the overall model that have not yet been explored, and possibly do not need to be explored, thus limiting invocations to local diagnosers. Local diagnosers explain blames and validate symptoms by means of an EXTEND operation, which provides extensions to partial assignments by assigning values to relevant unassigned variables; we will characterize the operation in section IV-C. Thus the partial assignments we will consider will assign *ok*/*ab* values to interface variables and *ok*/*fail* modes to internal activities.

B. A protocol for the global diagnoser *D*

During a diagnostic session, ***D*** keeps track of the progress by means of a list H of current partial assignments. Values are only assigned by local diagnosers, thus ***D*** becomes aware of the existence of a variable x only when a local diagnoser assigns a value to it. We will denote with $\alpha(x)$ the value of variable x in assignment α. We will write $\alpha(x) = *$ to denote that α does not assign any value to x.

For each assignment $\alpha \in H$ and for every interface variable x such that $\alpha(x) \neq *$ we assume that the identities of the sender $\mathsf{SND}(x)$ and the receiver $\mathsf{RCV}(x)$ of the messages where x is specified are known to ***D***: one is the local diagnoser A_i who first assigned a value to x, the identity of the other is provided by A_i itself. Notice that the receiver and sender of a message only need to be known at run-time. Moreover, ***D*** associates with each $\alpha \in H$ a list $\mathbf{L}_\alpha$ of local diagnosers that should extend α.

Given a partial assignment $\alpha \in H$ we denote by $\alpha(M_i)$ its restriction to interface variables and behaviour modes of M_i, and by $\alpha(\overline{M_i})$ its restriction to interface variables and behaviour modes *not* in M_i.

Local EXTEND operations work on partial assignments restricted to the local model they are invoked on. EXTEND will be characterized precisely in the following section; for now it suffices to know that, for each $\alpha(M_i)$ it receives in input, it returns a set of extensions $\mathbf{Ext}(\alpha(M_i))$ which relate values assigned in $\alpha(M_i)$ to values of other interface variables of M_i or to behaviour modes of activities in M_i; if the set of extensions is empty the assignment is considered to be *rejected*, because (as we will see in the next section) this means that the assignment is inconsistent with M_i and/or observations performed by its local diagnoser. The diagnostic process is started by a local diagnoser which is awakened by an alarm, and calls EXTEND to explain it. The result is provided to ***D*** as the initial value for H. ***D*** then executes a loop with the following steps.

Step 1: select the next request to a local diagnoser A_i. ***D*** finds a local diagnoser A_i that belongs to $\mathbf{L}_\alpha$ for some $\alpha \in H$; if there is none, exits the loop. From the point of view of correctness, how the choice is performed is ininfluent. In section V we will discuss policies.

Step 2: invoke EXTEND on A_i. If A_i has never been invoked before in this diagnostic process, then the input to EXTEND is $\{\alpha(M_i) \mid \alpha \in H\}$ (that is, the restrictions to M_i of the whole set H). Otherwise the input is the set of assignments $\{\alpha(M_i) \mid \alpha \in H$ and $A_i \in \mathbf{L}_\alpha\}$ (that is, the restrictions to M_i of those assignments that have changed from the last invocation).

Step 3: update H and the $\mathbf{L}_\alpha$ lists. This receives the output of EXTEND from A_i. For each $\alpha(M_i)$ in input, EXTEND has returned a set $\mathbf{Ext}(\alpha(M_i))$ of extensions. Then α is *replaced* in H by the set of assignments

$$\{\beta \mid \beta = \alpha(\overline{M_i}) \cup \gamma \text{ and } \gamma \in \mathbf{Ext}(\alpha(M_i))\}.$$

This implies that rejected assignments, having no extensions, are removed from H. For each assignment $\beta = \alpha(\overline{M_i}) \cup \gamma$ added in this way $\mathbf{L}_\beta$ is built as follows:

- For each $j \neq i$, if $A_j \in \mathbf{L}_\alpha$ then $A_j \in \mathbf{L}_\beta$;
- If there is an interface variable x such that $\mathsf{RCV}(x) = A_i$, $\alpha(x) = *$ and $\beta(x) = ab$ then $\mathsf{SND}(x) \in \mathbf{L}_\beta$. Intuitively, if A_i has blamed A_j for an abnormal value on its inputs, then A_j is asked to give an explanation.
- If there is an interface variable y such that $\mathsf{SND}(y) = A_i$, $\alpha(y) = *$ and $\beta(y) \neq *$ then $\mathsf{RCV}(y) \in \mathbf{L}_\beta$. Intuitively, if A_i has predicted a symptom for an output sent to A_j, then A_j is asked to validate it.

Notice that the diagnostic process terminates: new requests for EXTEND are generated only if assignments are properly extended, but assignments cannot be extended indefinitely.

At the end of the diagnostic process we can extract minimal consistency- based diagnoses from H as follows. We associate a diagnosis $\Delta(\alpha)$ to every $\alpha \in H$:

$$\Delta(\alpha) = \{x \mid x \text{ is an internal activity and } \alpha(x) = \textit{failed}\}$$

It can be proved that, if EXTEND behaves as defined in the next section, the set $\{\Delta(\alpha) \mid \alpha \in H\}$ contains all the minimal diagnoses for the observations provided during the process.

C. A characterization of local diagnosers

As described in the previous sections, the input to EXTEND is a set of partial assignments of *ok*/*ab* values to interface variables in M_i and of *ok*/*fail* modes to internal activities. A local diagnoser A_i regards α as an assignment to *all of its variables* and behaviour modes, although internal variables are all unassigned. The output of EXTEND is a set of extensions $\mathbf{Ext}(\alpha)$ for every assignment α received in input. Given an extended assignment β computed internally, EXTEND only returns its restriction $\textit{pub}(\beta)$ to public variables, which, as explained before, in this section we assume to be interface variables and behaviour modes of internal activities.

Each local diagnoser should extend partial assignments so that unassigned variables are only those that do not provide relevant information with respect to the current diagnostic process. The notion of *admissibility* of an assignment captures this idea: an assignment is *admissible* in a given model if it does not allow to infer anything more than the model alone on unassigned variables.

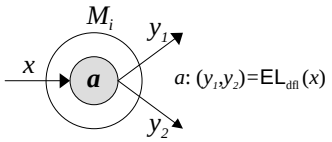

Fig. 3. A simple model M_i

Def. Let us denote by $\mathsf{DOM}(\alpha)$ the set of all variables x in a given model such that $\alpha(x) \neq *$, and by $\overline{\mathsf{DOM}}(\alpha)$ the set of unassigned variables. We say that an assignment α is *admissible* in M_i if *(i)* it is consistent with M_i and *(ii)* the restriction of $M_i \cup \alpha$ to variables in $\overline{\mathsf{DOM}}(\alpha)$ is equivalent to the restriction of M_i alone to $\overline{\mathsf{DOM}}(\alpha)$: $(M_i \cup \alpha) \mid_{\overline{\mathsf{DOM}}(\alpha)} \equiv M_i \mid_{\overline{\mathsf{DOM}}(\alpha)}$.

Requirement *(i)* (consistency) is actually implied by requirement *(ii)* for all but total assignments, for which $\overline{\mathsf{DOM}}(\alpha)$ is empty. As an example, let us look at the simple model M_i in figure 3, where we assume that activity a is modelled with a single $\mathsf{EL}_{\mathsf{dfl}}$ block, i.e. its model is the default EL model in section III. Let us consider the following partial assignments:

	a	x	y_1	y_2
α_1	$*$	$*$	$*$	*ab*
α_2	*ok*	*ok*	$*$	*ab*
α_3	*fail*	$*$	$*$	*ab*

Assignment α_1 is consistent with M_i but it is not admissible: in fact, M_i is consistent with a, x and y_1 being all *ok*, while $M_i \cup \alpha$ it is not, since when both a and x are *ok* also y_1 and y_2 must be ok. For the same reason, assignment α_2 is both inconsistent ad unadmissible wrt M_i. On the contrary, α_3 is admissible: in fact, it is consistent with all combinations of values for x and y_1.

Given an input set S of partial assignments, for each $\alpha \in S$, EXTEND computes a (possibly empty) set of extensions $\mathbf{Ext}(\alpha)$, defined as follows:

Def. Let A_i be a local diagnoser with model M_i, and let α be a partial assignment received by A_i as input to an EXTEND operation. Let moreover ω denote the assignment corresponding to internal observations (if any). The set $\mathbf{Ext}(\alpha)$ computed by EXTEND is the set of assignments:

$$\{\textit{pub}(\gamma) \mid \gamma \text{ is a minimal admissible extension of } \alpha \cup \omega\}$$

Let us consider again the example of figure 3, and let us consider the assignment α_1 mentioned above. We have $\mathbf{Ext}(\alpha_1) = \{\gamma_1, \gamma_2\}$ where:

	a	x	y_1	y_2
γ_1	*fail*	$*$	$*$	*ab*
γ_2	$*$	*ab*	$*$	*ab*

In this case, all possible extensions of γ_1 and γ_2 are admissible in the model. However, this is not true in general: an admissible assignment may have extensions that are inconsistent in the model. For example, the empty assignments is always admissible in any model.

Notice that EXTEND performs both a *consistency-based* explanation and a *consistency-based prediction*. Given an input assignment α, an observations assignment ω and a minimal admissible extension γ of $\alpha \cup \omega$, we have that:

- newly-assigned values in γ to input variables or behaviour modes can be seen as *explanations* of observations or output values assigned in α;
- newly-assigned values in γ to output variables can be seen as additional symptoms *predicted* by the above mentioned explanations.

V. Conclusions

In this paper we proposed a partially distributed *model-based* approach to diagnosis of complex Web Services. Web Services are modelled in a *component-oriented* fashion, as it is common in model-based diagnosis [4]; the term *component* in our case refers to internal service activities, which are the smallest diagnosable units. For individual activities we adopted *grey box* models: that is we do not model the internal behaviour of an activity, but only the correlation between its inputs and outputs. From this information we can infer how the correct/incorrect status of input parameters and of the activity itself affects the correct/incorrect status of output parameters.

In this sense our models are similar to those in [11]. The focus of [11] is not however on the diagnosis of composed Web Services, with its specific requirements on distribution of knowledge and reasoning. Their approach is purely distributed; in the context of Web Services, we motivated the adoption of a global diagnostic service for the composed service, which allows to reduce the communication flow between services. Moreover, [11] makes some restrictive assumptions on models. Another advantage of our approach is that makes selected predictions for discriminating candidates, but, by exploiting partial assignments, it avoids investigating those parts of the model that are not directly involved by blames or predicted symptoms.

A decentralized approach to diagnosis has been proposed in [10]. The application (telecommunication networks) is significantly different from ours, posing a very different problem. In our case, an alarm may be raised in a point that is far away from the failure source. In their case, a failure causes a chain of alarms, the first of which points to the failure source. However, due to the distributed nature of the network, the order in which alarms are received is not the same in which they are raised, thus the problem of finding the failure source.

A similar approach has been proposed for component-based software in [3], where chains of software exceptions are considered instead of alarms. Although the field of application is close to Web Services, the analysed problem remains different from the one tackled in this paper. Moreover, software components are modelled in black-box fashion, considering only their alarm-raising capability and not the correlations between input and output parameters.

In [6] Web Services are modeled in DAML-S, a Semantic Web ontology with a situation calculus semantics; the model is translated to Petri Nets for simulation and verification. Due to the different goals, their models provide a different abstraction of the Web Services with respect to the models proposed in this paper, with different implications from the computational point of view: for example, our models do not require reasoning on state changes. In principle, simulation/verification and diagnosis of systems (including software systems) could be based on a unified modeling approach.

Before such a goal can be pursued for Web Services, some more computational issues can be developed for the diagnosis approach in this paper. First of all, we did not specify which strategy $\boldsymbol{D}$ exploits in order to schedule EXTEND invocations on local diagnosers. Such a strategy would strongly depend on whether $\boldsymbol{D}$ knows in advance something about the interaction between the composed Web Services. In fact, as we noticed in Section II, the diagnostic framework we define does not make any assumption on how the services coordinate or are coordinated. This implies that initially the global diagnostic service has no information about the composed services and their coordination paths. Several approaches to coordination have been proposed in the Web Service community; e.g., cooperation either based on a workflow orchestrated by a service or based on intelligent invocation strategies relying on rich Semantic Web descriptions of service specifications. The availability of information about the network of cooperation between services or about semantic specifications of services could be used to focus the diagnostic process, and to define scheduling policies for the invocations to local diagnosers.

As to local diagnosers, we proposed a characterization of their operations (which, like most diagnostic tasks, can be computationally expensive in the worst case) without a specific algorithm. Precompilation and approximation techniques can be used to achieve diagnostic results efficiently for at least some classes of models: in particular, using templates and their default models should allow to use precompiled results.

References

[1] G. Alonso, F. Casati, H. Kuno, and V. Machiraju. *Web Services - Concepts, architectures and applications*. Springer, 2004.

[2] T. Andrews, et al. Business Process Execution Language for Web Services version 1.1. http://www-106.ibm.com/developerworks/webservices/library/ws-bpel/, 2003.

[3] I. Grosclaude. Model-based monitoring of component-based software systems. In *Proc. 15th Int. Work. on Principles of Diagnosis*, pages 155–160, 2004.

[4] W. Hamscher, L. Console, and J. de Kleer, editors. *Readings in Model-Based Diagnosis*. Morgan Kaufmann, 1992.

[5] S. McIlraith, T.C. Son, and H. Zeng. Semantic Web Services. *IEEE Intelligent Systems*, 16(2):46–53, 2001.

[6] S. Narayanan and S. McIlraith. Simulation, verification and automated composition of web services. In *Proc. 11th Int. WWW Conf.*, 2002.

[7] OWL Services Coalition. OWL-S: Semantic Markup for Web Services. http://www.daml.org/services/owl-s/1.1B/owl-s/owl-s.html, 2004.

[8] M. Paolucci, K. Sycara, T. Nishimura, and N. Srinivasan. Toward a Semantic Web e-commerce. In *Proc. of 6th Int. Conf. on Business Information Systems (BIS'2003)*, Colorado Springs, Colorado, 2003.

[9] M.P. Papazoglou and D. Georgakopoulos, editors. *Service-Oriented Computing*, volume 46. Communications of the ACM, 2003.

[10] Y. Pencolé and M.-O. Cordier. A formal framework for the decentralised diagnosis of large scale discrete event systems and its application to telecommunication networks. *Artificial Intelligence*, 2005.

[11] N. Roos, A. ten Teije, and C. Witteveen. A protocol for multi-agent diagnosis with spatially distributed knowledge. In J. Rosenschein and M. Wooldridge, editors, *2nd Int. Joint Conf. on Autonomous Agents and Multi-Agent Systems (AAMAS-2003)*, 2003. ACM.

[12] W. van der Aalst and K. van Hee. *Workflow Management - Models, Methods, and Systems*. The MIT Press, 2002.

[13] W3C. Web Services Definition Language Version 2.0. http://www.w3.org/TR/wsdl20/, 2004.