PROCESS ORIENTED DISCOVERY OF BUSINESS PARTNERS

Axel Martens
IBM T. J. Watson Research Center
Component Systems Group

Keywords: Business Process Modeling, Web Service, BPEL4WS, UDDI, Process Model Matchmaking, Petri nets.

Abstract: Emerging technologies and industrial standards in the field of *Web services* enable a much faster and easier cooperation of distributed partners. With the increasing number of enterprises that offer specific functionality in terms of Web services, *discovery* of matching partners becomes a serious issue. At the moment, discovery of Web services generally is based on meta-information (e. g. name, business category) and some technical aspects (e. g. interface, protocols). But, this selection might be to coarse grained for dynamic application integration, and there is much more information available. This paper describes a method to discover business partners based on the comparison of their *behavior* – specified in terms of their published Web service process models.

1 INTRODUCTION

To an increasing extend business processes are crossing the borders of individual enterprises. This requires the integration of the underlying business applications in a reliable but flexible manner. *Web services* (Alonso et al., 2002) provide a stack of closely related technologies to cover the heterogeneity and distribution of such processes underneath a homogenous concept of components and composition. Each self-contained work item or sub-process is wrapped up and encapsulated as a Web service – a reusable component with a standardized interface. The distributed application arises from composition of several Web services that interact via existing protocols.

1.1 The scenario

In spit of all technical specifications, Web services are not just another programming paradigm. They are the core piece of the *service oriented architecture* SOA (Alonso et al., 2002) – a philosophy of transforming business functionality into an article of trade, which is available through a network, and which can be discovered and integrated into an distributed application on demand. To provide necessary background information, Figure 1 shows the involved actors and their activities. For one given Web service each participant plays the role of a *provider*, a *requestor* or a *broker*.

The *service provider* offers a Web service to potential users by publishing (1) its description in a repository, called UDDI (Bellwood et al., 2002). The Web service description consists of several parts. Some parts cover non-operational information (e. g. name, address, business category, policies). This paper focuses of the technical parts, which cover, at the moment, basically the Web service *interface description*, defined in terms of WSDL (Christensen et al., 2001). As this paper will show, the Web service *process model* should be include, as well – defined in terms of the Business Process Execution Language for Weg Services BPEL4WS (Andrews et al., 2002).

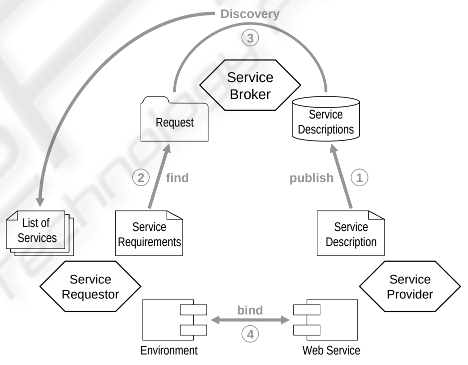

Figure 1: Service Oriented Architecture (SOA)

The *service requestor* wants to integrate a new Web service into its application – which is called *environment*. Because the resulting composed system should perform a desired task error free, the requestor demands certain properties from the Web service to find (2). This paper presents an intuitive way to formulate those requirements in terms of a process model.

The *service broker* manages a repository of published Web service description. His task is to discover (3) those services, which meet the needs of the service requestor. In general, there might be more

Martens A. (2005).
PROCESS ORIENTED DISCOVERY OF BUSINESS PARTNERS.
In *Proceedings of the Seventh International Conference on Enterprise Information Systems*, pages 57-64
DOI: 10.5220/0002527300570064
Copyright © SciTePress

than one matching service. Hence, the broker presents the requestor a list of candidates. Finally, the requestor chooses one service and integrates it into his environment. This activity is called binding (4). Due to the presented discovery mechanisms, compatibility on the process level between the environment of the requestor and any discovered service is guaranteed.

1.2 The approach

Apparently, the quality of the matching between a Web service description and the requested properties has a strong influence on the convenience and success of the binding. At the moment, matching of technical aspects is limited to comparison of the interface description, which might not gain sufficient precision. It is possible that the internal structure of a discovered service does not meet the customers needs although it has the required interface. For example, a service requestor is searching a shopping service that delivers an ordered product and requires payment afterwards. Focused on the interface only (sending product, receiving payment), the discovery mechanisms might point to a service that requires payment first. Obviously, there is a mismatch between the desired and the discovered service. In that sense, the matching criterion is not adequate.

Instead of considering the Web service interface description only, this paper presents a matching mechanism that additionally compares the behavior of Web services – which is defined by the *Web service process model*. To enable this comparison, three activities of the service oriented architecture has to be enhanced as follows: First, the service provider publishes a BPEL process model as an additional part of the service description. Second, the service requestor models his requirements in terms of an (*abstract*) BPEL process model as well, similarly to the *Query-By-Example* approach of database theory. And third, the service broker compares the behavior of the published and requested process models – in addition to the established discovery mechanisms – which increases the precision dramatically.

This paper explains, how already existing efficiently executable formal methods can be used to perform the necessary step. Therefore, Section 2 introduces the Web service process modeling language BPEL4WS by help of an example and discusses the matchmaking requirements. Section 3 presents a formal representation of the Web service behaviors and derives the property of *simulation* between two process models. This properties the foundation of the proposed matchmaking mechanism. In Section 4, the integration into the service oriented architecture is demonstrated in different ways, and more advanced use cases are discussed. Finally, the conclusion relates this approach to other published research efforts.

2 PROCESS MODELING

The *Business Process Execution Language for Web Services* BPEL4WS (Andrews et al., 2002) is in the very act of becoming the industrial standard for modeling Web service based business processes. Hence, this language is used in this paper to specify, implement and compare the behavior of Web services. Figures 2 and 3 show two BPEL models of two different Web services processing incoming orders.

2.1 Modeling with BPEL4WS

Basically, a BPEL process model consists of two different kinds of activities: *Basic activities* are used to communicate to the outside e. g. receive order, confirm order in Figure 2), to manipulate data or to interfere with the control flow, e. g. by signaling faults (not present in the examples). *Structured activities* aggregate other activities and therefore, they are used to build the control structures of the process: A sequence is the simplest structured activity. Alternative branches can be either based on data (switch on order status in Figure 2) or based incoming on messages (pick customer's message in Figure 3). The parallel execution of activities might be synchronized by directed links (cf. Figure 8). Moreover, BPEL4WS allows the repeated of activities, which does not appear in the chosen examples.

Let's have a closer look on order process 1, shown in Figure 2. Its structure is quite simple: First, this service receives an order from a customer. Then, the service chooses one out of two alternative branches. Either the order is accepted and a confirmation is send, or the order is rejected (e. g. because the product is out of stock). In that case, the customer gets the shop's special offers instead. It is not specified, how the decision between these two branches is made.

Let's assume, this process model was specified by the service requestor, and therefore, it is called *specification*. According to the service oriented architecture, the requestor is looking for a Web service that could be bind to his environment, such that the resulting composed system performs the desired task error free. In other words: The environment and the discovered Web service have to be *compatible* – a property that could be proven effectively as described in (Martens, 2005a).

Of course, the BPEL model of order process 1 was designed such that it specifies the required functionality, and it is compatible to the requestor's environment. Hence, each Web service that is to be discovered should behave very much like this process model. If this is the case, the Web service is called *implementation*. To verify whether a discovered Web service is a valid implementation of a specification, in the next section the notion of *simulation* between their

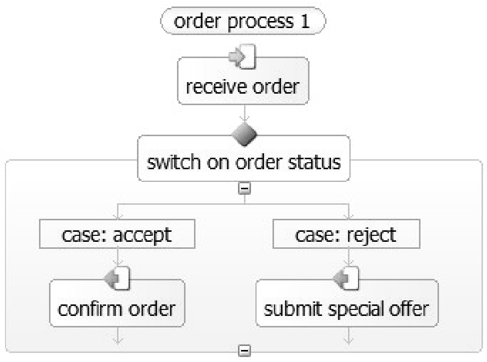

Figure 2: The requested Web service

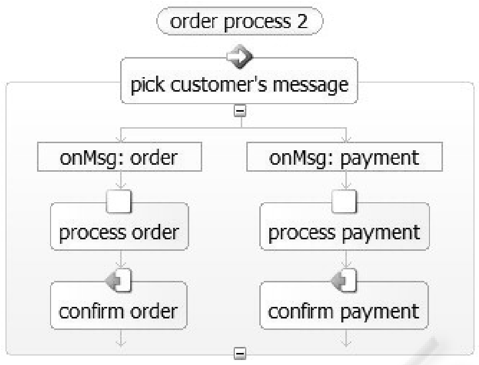

Figure 3: A published Web Service

behavioral models is defined. But before going into the precise formalism, the overall intuition of simulation shall be explained by comparing the specification with the BPEL model of order process 2.

2.2 Matching process models

Figure 2 shows the process model of a published Web service. The structure of this process model differs a lot from the previously explained order process 1, and so does the behavior. Similarly to the specification, order process 2 is able to receive an order from the customer right at the beginning. But, it also can receive the payment. In case of an order, order process 2 performs an internal activity and sends always a confirmation. It does neither have the possibility to reject an order nor to send some special offers instead.

In spite of all differences, it is reasonable to call order process 2 a valid implementation of order process 1. To prove this statement, the *compatible* environments of order process 1 have to be considered. Each of those environments always will send an order to the service, because this is the only message that could be received by order process 1. This message could be received by order process 2,
as well. After sending the order, an compatible environment has to be able to receive a confirmation, because order process 1 could decide to send this message. Because this is the only response order process 2 is able to send, the environment is perfectly compatible to order process 2. Moreover, such an environment is not able to see any difference between those two Web services.

This example already poses some requirements to the notion of simulation. The implementing service has to be able to accept *at least* those messages, the specifying service can accept, and the implementing service is allowed to produce *at most* those messages, the specifying service can produce – always in respect of the current state, i. e. the history of communication. Nevertheless, the implementing service does not need to cover all behavior of the specifying service, and the implementing service might provide additional behavior. The next section pinpoints the common pattern. At least, there must be a subset of the specifying service's behavior that is performed in the same manner by the implementing service. In the chosen example, the sequence of receiving and confirming an order is such a subset.

3 COMPARING BEHAVIOR

The presented approach to Web service discovery is based on the comparison of the *behavior* – specified in terms of their published BPEL process models. But, the semantics of BPEL4WS so far is defined only by English prose or encoded into middleware components, more ore less accurately. To reason precisely about the current state and activated actions, and to prove properties like the simulation of two models, a formal and explicit semantics of the language is needed. This paper is based on a formal Petri net semantics (Schmidt and Stahl, 2004) for BPEL4WS, which enables the automatic generation of an explicit behavioral model. This section gives a short overview on the algorithmic steps related to the examples, and refers to detailed explanations in other publications.

3.1 BPEL transformation

Petri nets are a well established method for modeling and analyzing (cross-organizational) business processes (van der Aalst, 1998a; van der Aalst, 1998b). Beside the already mentioned BPEL semantics, other recent research projects apply Petri nets to Web Services (Hamadi and Benatallah, 2003).

Basically, a *Petri net* consists of a set of *transitions* (boxes), a set of *places* (ellipses), and a *flow relation* (arcs) (Reisig, 1985). A transition represents a dynamic element, i. e. an activity of a business process

(e. g. receive order). A place represents a static element, i. e. the causality between activities or a message channel (e. g. order). The state of a Petri net is represented by black *tokens* distributed over the places. While transforming a BPEL process into a Petri net, each element of the source language is represented by a modular Petri net pattern. Hence, the process structure is still visible in the resulting net.

In contrast to the original BPEL process, the resulting Petri net abstracts from data aspects, i. e. each data driven decision is mapped into a non-deterministic choice. This is a usual procedure in the field of computer aided verification. On the one hand, a model without data has the disadvantage of less precision (i. e. a more general behavior). But on the other hand, this enables analysis methods yielding important results that are not applicable to models with full expressive power of arbitrary data objects.

Because of the possible loss of information, the implemented transformation provides the feature of user interaction. On specific points that a crucial for the process' behavior, the user may add structure to the Petri net. Thereby, data dependencies are transformed into control dependencies (cf. Section 4.2).

Figure 4 shows a screenshot of the Web service analysis tool WOMBAT4WS (WOMBAT4WS, 2003,). In the left half, it shows the generated Petri net model of the order process 1 (cf. Figure 2). Based on the transformation into Petri net, WOMBAT4WS provides several analysis methods for BPEL processes: the check of *usability* of a given Web services (Martens, 2004), the check of *compatibility* (Martens, 2003) of two Web services, the *automatic generation* of an abstract process model for a given Web service, and the check of *simulation* between two Web services (Martens, 2005b) – which is the base for presented discovery mechanisms. The following two paragraphs reflect the idea of simulation and its verification.

3.2 Behavioral model

In this approach, the comparison of the behavior is not defined on the generated Petri net models directly. This is because those models reflect the entire structure of the corresponding BPEL processes, and therefore they might contain details that have no direct impact on the communication with the environment. Instead, an explicit representation of the externally visible behavior is derived – called the *communication graph* (c-graph) of a BPEL process. A c-graph is a finite deterministic automaton. In the right half, Figure 4 shows the c-graph of the order process 1.

The nodes of a c-graph are divided into two classes: *visible nodes* (white ellipses) and *hidden nodes* (black dots), and its edges are labeled with messages either received (e. g. order) or send (e. g. confirm) by the process. A visible node represents a state in which

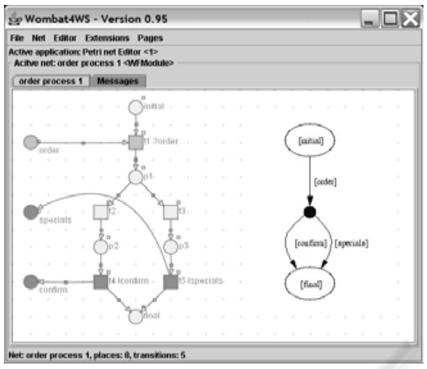

Figure 4: Transformation into a behavioral model

the process is either waiting for an *input* message or already terminated. Hence, the c-graph's initial node and its final nodes are visible. A hidden node represents a state in which the process is able to produce an *output* message. Each visible node is followed by a hidden node and vice versa. A path from the initial node to the final node represents a communication sequence between the process and its environment.

Because a BPEL process is able to consume messages from the environment and to produce answers depending on its internal state, the state of the process is important – simple message traces do not provide the same precision (cf. (van Glabbeek, 1990)). But, an environment, which wants to use this process, has no *explicit* information on its internal state. Instead, an environment can derive information from the process model by considering the history of communication, only. Consequently, an environment might gain *implicit* information. Exactly that kind of information is represented within the c-graph of the BPEL process.

In general, the c-graph of a BPEL process may contain multiple leaf nodes. But in each c-graph, there is at most one leaf node, which is labeled with the proper final state of the Petri net model. All other leaf nodes contain at least one state, where there are messages left, or which marks a deadlock state of the Petri net. If an environment was communicating with the process according to a path towards such a leaf node, the composed system of the process and the environment will end up in an erroneous situation. The elimination of all such erroneous sequences yields a (possibly empty) subgraph that can be regarded as an user manual of the module – called the *usability graph* (u-graph). A c-graph may contain several u-graphs, in general. In all chosen examples, the whole c-graph also is one u-graphs of the corresponding BPEL process. The precise, mathematical definition of all notions mentioned above and a discussion on the complexity of the generating algorithms is presented in (Martens, 2005a).

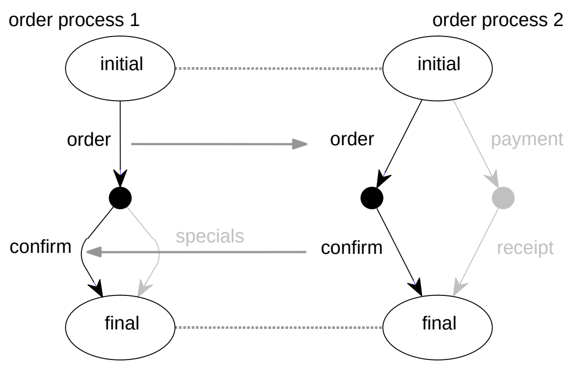

Figure 5: Simulation between c-graphs

3.3 Defining simulation

Intuitively, one BPEL process simulates another BPEL process if in a *comparable situation* the first process *behaves like* the second process. There are countless approaches published dealing with the comparison of behavior in terms of simulation or equivalence, respectively. This variety results from different interpretations of the terms "comparable situation" and "behave like". To find the appropriate terms, first of all, a semantic definition of simulation should be derived from the field of application. In a second step, a proving method can be developed.

Definition 3.1 (Simulation/Equivalence).
A BPEL process A *simulates* a BPEL process B if each compatible environment of process B is an compatible environment of process A, too. Two BPEL processes A and B are called *equivalent*, if process A simulates process B and vice versa. $\star$

This definition meets the requirements of an discovery mechanism in the service oriented architecture. A service requestor has an environment, and he defines an abstract process such that it is compatible to this environment. Hence, if the service broker points him to a published service that simulates the abstract service, the compatibility with the requestor's environment is guaranteed. To illustrate the verification of this property, Figure 5 shows the c-graphs of both order processes introduced in Section 2. As mentioned before, order process 2 is a valid implementation of order process 1. Hence, in Figure 5, the pattern of simulation can be seen. The order process 2 can accept at least those messages, order process 1 can accept (message order). Once the message was received, order process 2 produces at most those messages, order process 1 can produce (message confirm). To emphasize the connection to the process discovery, P refers to the *published* service and R refers to the *requested* service.

Activated input: Given a visible node of an u-graph, a bag of messages is an activated input in this node if the node has an outgoing edge that is labeled with this bag of messages.

Communication step: Given a visible node v of a c-graph and a bag of messages m, the tuple (v, i, o, v') is a *possible communication step*, if there is an edge in the c-graph from v to a hidden node h that is labeled with the a bag of messages i, there is an edge from h to a visible node v' that is labeled with a bag of messages o, and m greater or equal to i. Two steps are called *similar* if the labels are identically.

Definition 3.2 (Simulation).
A c-graph $C(P)$ *simulates* a c-graph $C(R)$ if both graphs contain a non-empty u-graph, and the root node of $C(P)$ simulates the root node of $C(R)$.
A visible node p of $C(P)$ *simulates* a visible node r of $C(R)$ if the following requirements are fulfilled:

(i) For each activated input in r and for each possible communication step in p (with this input), there is a similar communication step in r and the target node r' simulates the target node p'.
(ii) If r is a leaf node, then p is a leaf node, too.
(iii) If i is an activated input in p, and i is a subset of any activated input in r, then i is an activated input in r, too. $\star$

Requirement (i) has been motivated before: If a BPEL process P shall simulate a BPEL process R, then P has to accept *at least* those messages R accepts, and P has to produce *at most* those messages R produces. The proper simulation of terminal states is guaranteed by requirement (ii). Finally, the process P must not respond to a smaller bag of messages than R does, e. g. sending a product without receiving the complete payment. Such behavior can yield to a deadlock, e. g. some customers might refuse to complete the payment after receiving the product.

Regarding to the examples of Figure 5, it is easy to prove that the right u-graph simulates the left u-graph. The presented definition of simulation between c-graph is a sufficient criterion for simulation between BPEL processes. Hence, the simulation relation between order process 2 and order process 1 is proven. In (Martens, 2005b), this simplified definition is extended to communicating pathes instead of communication steps. With this extension, the following theorem holds:

Theorem 3.1 (Simulation).
A BPEL process P simulates a BPEL process R iff the c-graph $C(P)$ simulates the c-graph $C(R)$. $\star$

As a result, *simulation* of BPEL processes can be decided effectively. The entire proof, a detailed discussion, and precise definition of all notions can be found in (Martens, 2005b). Now, all prerequisites are given to build the simulation analysis into the discovery mechanism.

4 DISCOVERY

The simulation analysis build the formal backbone of the presented discovery mechanism. To gain an integrated solution, some design issues have to be decided: Who is responsible for the generation of the behavioral model? When will the simulation check take place? What changes to the existing architecture has to be made? This section sketches two different implementations and one combined approach. Supplementary, it deals with the problem of data dependencies that are crucial to the process' behavior.

4.1 Discovery architectures

Given an extensive repository of published Web service, the first step of selecting possible candidates often is based on non-operational information: A service requestor is looking for a partner that runs his business in a certain domain, that is located in a certain area or that has a certain amount of expertise (references) in providing the required functionality. This discovery step is either implemented by help of a hierarchically structured catalogue (i. e. taxonomy (Boubez and Clément, 2002)) or through semantic net of notions, relations and dependencies (i. e. ontology (Paolucci et al., 2002; Web-Ontology Working Group, 2004)). Assuming an efficient way of performing step one is available, this paper focusses on the second step – based on the operational Web service description.

Server based approach Figure 6 visualizes the first – so called *server based approach*. The service provider has created a BPEL process model of his Web service. He transforms the process model into a Petri net and generates the *behavioral model* (1a), i. e. the c-graph of the BPEL process. Built into an end-user process modeling tool, the transformation and generation are performed on the backend in such a way that the user only has to deal with BPEL process models. The service provider publishes the behavioral model (1b) – which also contains the interface description and refers to the actual service implementation – into the repository.

The service requestor behaves almost similarly. But, his BPEL process models describes the required behavior. He also generates the behavioral model (2a) and submits it, together with the non-operational requirements, to the service broker (2b). After the service broker has performed the first discovery step (see

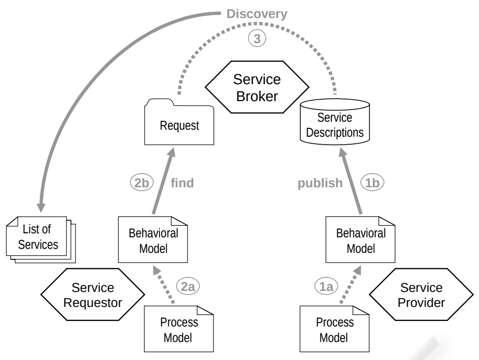

Figure 6: A server based approach

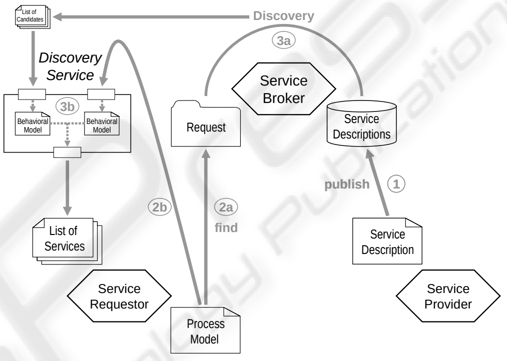

Figure 7: A service based approach

above), he compares the interfaces of the remaining published services with the request. If a published service has a matching interface, the service broker checks whether it simulates the requested behavior. Only if this check yields true, the published service is appended to the list of services that is presented to the requestor.

In this approach, both process modeling parties (the provider and the requestor) have direct influence on the generation of the behavioral model. This fact that will be appreciated in the next section. But this approach poses some none trivial challenges: It requires mayor changes to the UDDI specification – which has to be done anyhow, because the representation of BPEL4WS in the repository is not solved sufficiently, yet. Moreover, the issues of scalability and performance are not sufficiently examined yet.

Service based approach Figure 7 visualizes an alternative – so called *service based approach*. In this approach, the service provider just publishes his Web service description (1), as required in the service oriented architecture (cf. Figure 1). The discovery runs into two steps: First the service requestor builds a standardized request from his abstract process model and submits it to the broker (2a). The broker discovers a list of candidates (3a), basically based on the interface description. The requestor redirects this list together with his process model (2b) to an external *discovery service*. This service generates the behavioral model of the requested process and all selected candidates. Then, it performs the simulation analysis, such that a smaller list of services is finally presented to the requestor (3b). A mayor advantage of this approach is the relatively small changes to the established discovery mechanisms (only the requestor has to change his behavior). Moreover, this architecture provides a simulation check *on demand* and has the build-in feature of scalability.

Hybrid approach The discovery service has to generate the behavioral model of a published Web service without the possibility of interacting with the service provider. This might be a disadvantage of this approach in some cases, as explained in the following paragraph. To overcome this problem, a kind of hybrid approach might be considered: The service provider and the service requestor generate their behavioral models, as shown in the server based approach. The provider publishes this model supplementing the Web service description. The broker ignores the behavioral model while performing discovery, but includes it when returning the list of candidates. The requestor submits his behavioral model to the discovery service, as well. In result, the discovery service only has to perform the simulation check. Such an approach combines modeler's direct influence on the behavioral model generation and scalability – without mayor changes to existing discovery architectures.

4.2 Discovery challenges

Regarding to the chosen example, the objected precision of discovery is obtained only if the behavioral models of both processes reflect the behavior adequately. As mentioned already, the transformation from BPEL4WS to Petri nets abstracts form data aspects. In result, the arising Petri net has a more general behavior than the actual BPEL process. In most cases, this still is an adequate approximation. But for some process model, the behavior of generated Petri net might be too arbitrary. Figure 8 shows a third order process that has an area of parallel activities, partially synchronized.

The order process 3 shown in Figure 8 receive the order and contacts three contractors in parallel to check the availability of the requested product. In the published process model, those activities (call contractor..) are *empty* activities instead a actual *invoke* activities, because they do not communicate with the customer. Each of them has to outgo-

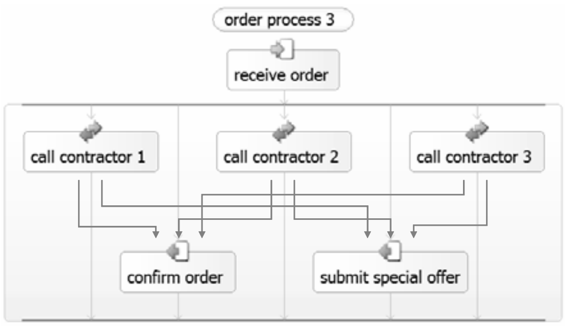

Figure 8: Another published Web service

ing links: If a contractor has the requested product in stock, the link towards the activity confirm order is set to true, and the other link is set to false. Otherwise, the values of the links are flipped. The activity confirm order is activated if at least one incoming link has the value true. The activity submit special offer is activated if all incoming links have the value true – i. e. no contractor has the requested product in stock.

Due to this structure, the process models an exclusive choice between the activities confirm order and submit special offer. But, because predicates on real data are used to specify the link values (called *transition condition*) and the join behavior (called *join condition*), a pure non-deterministic transformation into a Petri net would not yield an good approximation. Instead, the tool WOMBAT4WS asks the process modeler to choose a more appropriate pattern, e. g. a *xor-split* replacing the transition conditions and an *or-join* or an *and-join* replacing the join conditions. By help of pattern, each boolean combination can be expressed (cf. (Martens, 2005a) for more details). If the process modeler can't be contacted, the algorithm tries to choose a pattern automatically. Obviously this does not yield the same precision. But, current research on static program analysis aims to improve the quality of mapping BPEL4WS into Petri nets.

Once the data dependencies are transformed into control structures, the c-graph of order process 3 could be generated. It looks exactly like the c-graph of order process 1 (cf. Figure 5). Hence, order process 3 simulates order process 1, i. e. the service requestor who has specified order process 1 will be totally satisfied binding its environment to the order process 3. This result has not been achieved without interaction of the process modeler.

5 CONCLUSION

This paper presents a method to discover business partners based on the comparison between the required behavior and the behavior implemented by an offered service. This method is an extension to established discovery mechanisms that uses available information and existing formal methods to gain a higher level of precision. The presented approach is based on a formal Petri net semantic (Schmidt and Stahl, 2004) for the modeling language BPEL4WS. Consequently, the method is directly applicable to real world examples. The existing tool WOMBAT4WS proves it effectiveness (WOMBAT4WS, 2003,).

The current work was inspired by many other approaches, dealing with behavioral comparison of process models. Some of them also use Petri nets for process modeling (van der Aalst, 1998a; Hamadi and Benatallah, 2003). Those publications, of course, had influence on the current approach. But, none of them presents such a focussed view on a components externally visible behavior as the communication graph does. Concerning matchmaking of Web services, there are recent results published, which employ finite state automata to solve a similar problem (Wombacher et al., 2004). This approach seems to yield similar theoretical results, whereas the application to a real world modeling language is not provided, yet.

Currently, the simulation between process models is based on the assumption of one environment that uses all interfaces of the process. But, a BPEL process might interact with several partners, and therefore it is reasonable to have several abstract models of the same process. Each abstract model emphasizes only those interactions of one specific partner. In the next step, the notion of the communication graph and the definition of simulation will be adopted to meet the requirement of this scenario.

The presented algorithms is already prototypically implemented. The current work is focussed on improving the algorithms' efficiency by the application of *partial order reduction* techniques (Valmari, 1988; Schmidt, 2002). Moreover, up to a certain degree the integration of data aspects into the formalism is planned. Especially the dependencies between the content of incoming message and internal decisions made by the process are the focussed target. Applying technologies of static program analysis (e. g. *slicing* (Nielson et al., 1999)), it seems possible, to achieve a higher level of precision in mapping a given process model into a Petri net, without loosing the possibility of efficient analysis.

REFERENCES

Alonso, G., Casati, F., Kuno, H., and Machiraju, V. (2002). *Web Services.* Springer-Verlag.

Andrews et al. (2002). *BPEL4WS – Business Process Execution Language for Web Services.* OASIS, Standard proposal, Version 1.1.

Bellwood, T., Clément, L., and von Riegen, C. (2002). *UDDI – Universal Discovery, Description, and Integration.* UDDI.org, Version 3.0, Standard.

Boubez, T. and Clément, L. (2002). *UDDI Registry tModels.* Technical contribution, Version 2.03, OASIS.

Christensen, E., Curbera, F., Meredith, G., and Weerawarana, S. (2001). *WSDL – Web Services Description Language.* W3C, Standard, Version 1.1.

Hamadi, R. and Benatallah, B. (2003). *A Petri Net based Model for Web Service Composition.* In *Proc. of ADC 2003*. Australian Computer Society, Inc.

Martens, A. (2003). *On Compatibility of Web Services. Petri Net Newsletter*, 65:12–20.

Martens, A. (2004). *On Usability of Web Services.* In Calero, Diaz, and Piattini, editors, *Proc. of WISE 2003 Workshops*, Rome, Italy. IEEE Computer Society.

Martens, A. (2005a). *Analyzing Web Service based Business Processes.* In *Proc. of FASE'05*, Edinburgh, Scotland. Springer-Verlag, LNCS. to appear.

Martens, A. (2005b). *Consistency between Executable and Abstract Processes.* In *Proc. of IEEE EEE'05*, Hong Kong. IEEE Computer Society. to appear.

Nielson, F., Nielson, H. R., and Hankin, C. (1999). *Principles of Program Analysis.* Springer-Verlag.

Paolucci, M., Kawamura, T., Payne, T., and Sycara, K. (2002). *Importing the Semantic Web in UDDI.* In *Proc. of CAiSE 2002*, Toronto, Canada.

Reisig, W. (1985). *Petri Nets.* Springer-Verlag.

Schmidt, K. (2002). *Explicit State Space Verification.* Postdoctoral thesis, Humboldt-Universität zu Berlin.

Schmidt, K. and Stahl, C. (2004). *A Petri net semantic for BPEL4WS.* In *Proc. of 11th Workshop AWPN*. University Paderborn, Germany.

Valmari, A. (1988). *State Space Generation – Efficiency and Practicality.* PhD thesis, Tampere University.

van der Aalst, W. (1998a). *Modeling and Analyzing Interorganizational Workflows.* In *Proc. of CSD'98*. IEEE Computer Society.

van der Aalst, W. (1998b). The Application of Petri Nets to Workflow Management. *Journal of Circuits, Systems and Computers*, 8(1):21–66.

van Glabbeek, R. (1990). *The Linear Time - Branching Time Spectrum.* LNCS 458. Springer-Verlag.

Web-Ontology Working Group (2004). *Ontology Web Language for Web Services.* W3C, OWL-S 1.1 Release.

Wombacher, Fankhauser, Mahleko, and Neuhold (2004). *Matchmaking for Business Processes.* In *Proc. of IEEE EEE-04*. IEEE Computer Society.

WOMBAT4WS (2003). *Workflow Modeling and Business Analysis Toolkit for Web Services.* Tool hompage, `http://www.informatik.hu-berlin.de/top/wombat/`.