IPSI-PF: A Business Process Matchmaking Engine

Andreas Wombacher

Bendick Mahleko

Erich Neuhold

Fraunhofer Gesellschaft, Integrated Publication and Information Systems Institute,
64293 Darmstadt, Germany
firstname.lastname@ipsi.fraunhofer.de

Abstract

Success of web services mainly depends on the availability of tools facilitating usage of technology within the addressed B2B integration problems. One severe problem in loosely coupled systems is service discovery including a sufficient matchmaking definition. The concept for service discovery in web service architecture is UDDI providing limited querying functionality and not being capable to deal with the multiple dimensions of a service, like for example semantic, workflow, or Quality of Service aspects. The IPSI Process Finder (IPSI-PF) provides a matchmaking definition and an engine focusing on process aspects. In particular, the matchmaking engine realizes service discovery by extending the capabilities of UDDI to ease the integration with additional UDDI extensions addressing other service description dimensions.

1. Introduction

Web services and related technologies promise to facilitate the efficient execution of B2B e-commerce by integrating business applications across networks like the Internet. A lot of effort has been expended to define standards, e.g., to model business processes and describing interfaces (BPEL4WS, WSDL, etc.) as well as specifying the technical infrastructure for carrying out business transactions (e.g., SOAP, UDDI). If web services are to be as successful as predicted, tools that aid companies, organizations and individuals in their day-to-day work must be available. Applicability of web service technologies in real life scenarios will thus dependent to a large scale on the availability and usefulness of such tools, and what impact they have on the business community.

Web services are advertised as a technology for implementing loosely coupled business processes, that is a dynamic and flexible binding of services. To date, web services are used as stateless components accessible via a sin-

gle request-response remote procedure call (RPC). One reason for this limited usage is the missing support for searching and finding state-maintaining/complex web services.

Within this paper we present a tool that allows searching and finding service providers by an automated comparison of business processes for compatibility based on an existing UDDI infrastructure. Such a tool takes as input a business process description belonging to one party and returns references to service providers offering services that are compatible with the input business process. The idea is to allow companies and organizations to automatically determine from a business process viewpoint, whom they can do business with. Alternatively, if the organization is already aware of who their potential business partners are, the tool allows them to check whether or not they fulfill their requirement in terms of business process compatibility. Of course the assumption here is that a standard exists that describes what messages and roles are permitted. Such standards exist already like e.g., OTA, RosettaNet, IOTP, etc. We have implemented such a tool and the aim of this paper is to describe its implementation and how it can be used within the web service framework. We believe that it is the ubiquity of tools like this one that will make web services widely used in real life.

The paper is organized as follows: Section 2 describes an example sample application, Section 3 explains the used approach and Section 4 presents the architecture. Evaluation of the concept and the implementation behind **IPSI-PF** process matchmaking engine is done in Section 5, followed by a discussion on related work in Section 6. Summary and future work are presented in Section 7.

2. Sample Application

The exemplary scenario used for further discussion is a simple procurement workflow within a virtual enterprise incorporating a buyer, an accounting department, and a logistics department. The accounting department approves an order (*order* message) sent by a buyer and forwards the

order to the logistics department (*deliver* message) to deliver the requested goods. The logistics department confirms the receipt (*deliver_conf* message) and forwards it to the buyer extended by the expected deliver date and the parcel tracking number using the *delivery* message. Further, the buyer may perform parcel tracking (*get_status* and *status* messages) of the shipped goods, which is forwarded by the accounting department to the logistics department. The overall scenario is depicted in Figure 1.

Figure 1. Global Procurement Scenario

The scenario sketched above is used for explaining the concepts within the IPSI Process Finder (IPSI-PF) match-making engine. In the following the local workflow of the accounting department denoted according to the BPEL specification is described in more detail. To keep the example simple, the specification of the messages is neglected and the names of the messages are simplified. Concrete message types could be exemplary taken from the RosettaNet Partner Interface Processes (PIPs) 3A4 (Request Purchase Order), 3A7 (Notify of Purchase Order Update), 3B2 (Notify of Advanced Shipment).

```
<portType name="accounting">
  <operation name="orderOp">
    <input message="tns:order"/>
  </operation>
  <operation name="getStatusOp">
    <input message="tns:getStatus"/>
  </operation>
  <operation name="terminateOp">
    <input message="tns:terminate"/>
  </operation>
</portType>
<portType name="accountingCallback">
  <operation name="deliveryOp">
    <input message="tns:delivery"/>
  </operation>
  <operation name="statusOp">
    <input message="tns:status"/>
  </operation>
</portType>
<portType name="logistic">
  <operation name="outOfStockOp">
    <input message="tns:delivery"/>
  </operation>
  <operation name="deliver_confOp">
    <input message="tns:deliver_conf"/>
  </operation>
</portType>
<portType name="logisticCallback">
  <operation name="getStatusOp">
    <input message="tns:getStatus"/>
    <output message="tns:status"/>
  </operation>
  <operation name="deliverOp">
    <input message="tns:deliver"/>
  </operation>
  <operation name="terminateOp">
    <input message="tns:terminate"/>
  </operation>
</portType>
```

Figure 2. exemplary WSDL porttype definition

Within web service specification one or more messages specify an operation representing a potential message ex-

change. If a porttype contains only a single input, than the operation is asynchronous, otherwise it is synchronous. A porttype contains a set of operations provided by a service provider, which is specified in the corresponding WSDL files. Figure 2 gives an overview of the operations and the related messages used in the exemplary accounting department BPEL description, where the messages are labeled in accordance to the high level description of the scenario. The *accounting* and the *logistics* porttypes represent the operations provided by the corresponding department service, that is contain messages that are received by the accounting and logistics department respectively. Consequently, the *accountingCallback* and *logisticCallback* porttypes contain operations provided by the buyer and the logistics department respectively. All operations are asynchronous except the *getStatusOP* operation provided in the *logisticCallback* porttype, which is synchronous.

```
<process>
  ...
  <sequence name="accounting department process">
    <receive partnerLink="buyer" portType="tns:accounting"
      operation="orderOp" variable="order"/>
    <invoke partnerLink="logistic" portType="tns:logisticCallback"
      operation="deliverOp" inputVariable="order"/>
    <receive partnerLink="logistic" portType="tns:logistic"
      operation="deliver_confOp" inputVariable="order"/>
    <invoke partnerLink="buyer" portType="tns:accountingCallback"
      operation="deliveryOp" inputVariable="order"/>
    <while name="parcel tracking" condition="1=1">
      <pick>
        <onMessage partnerLink="buyer" portType="tns:accounting"
          operation="getStatusOp" variable="getStatus">
          <sequence>
            <invoke partnerLink="logistic" portType="tns:logisticCallback"
              operation="getStatusOp" .../>
            <invoke partnerLink="buyer" portType="tns:accountingCallback"
              operation="statusOp" inputVariable="status"/>
          </sequence>
        </onMessage>
        <onMessage partnerLink="buyer" portType="tns:accounting"
          operation="terminateOp" variable="terminate">
          <sequence>
            <invoke partnerLink="logistic" portType="tns:logisticCallback"
              operation="terminateOp" inputVariable="terminate"/>
            </sequence>
          </onMessage>
        </pick>
      </while>
    </sequence>
  </process>
```

Figure 3. BPEL notation of the accounting workflow

The description of the local workflow is based on these porttype definitions by directly referencing them. The accounting department local workflow denoted as BPEL process is depicted in Figure 3. The process starts by receiving an *order* message sent by the buyer, which is forwarded to the logistics department via a *deliver* message. The logistics department answers asynchronously with a *deliver_conf* message, which is forwarded by the accounting process to the buyer via a *delivery* message. Due to the fact that the buyer is allowed to do parcel tracking an undetermined number of times, the parcel tracking must be contained within a non-terminating loop. To enable a termination of

the accounting and logistics department processes a *termination* message initiated by the buyer and forwarded by the accounting to the logistics department terminates the corresponding processes. Alternatively, the accounting department may receive a *get_status* message sent by the buyer, which is forwarded by a synchronous invocation of the logistics process and reporting the status via a *status* message back to the buyer.

Based on this process description the aim of an accounting department is to find a logistics service provider, which is compatible with the own local workflow, that is finding a complex web service provider guaranteeing a successful business interaction. To illustrate the concepts of the IPSI Process Finder (IPSI-PF) the following exemplary query is issued by the accounting department: services are searched, which i) support the role of a logistics department, ii) are provided by a business entity located in Germany, and iii) are compatible with the own local workflow specified by a BPEL process and the corresponding WSDL document.

3. Approach

Analyzing the above query shows, that the first two constraints can already be served by a UDDI registry being part of the web service infrastructure. The last constraint can not be handled by a UDDI registry, because the required comparison operation is more complex than string comparison as provided by a UDDI registry. Due to this observation, the query is decomposed into a UDDI and a local workflow sub-query. The final query result is derived on behalf of the partial results by doing a natural join, where the business service key maintained by the UDDI and being unique for each service instance within the UDDI is used as a primary key. The local workflow sub-query uses the business service key as a foreign key.

3.1. UDDI Sub-query

The UDDI query is based on standard UDDI API, that is a *find_service* SOAP call [11]. The call related to the above example is depicted in Figure 4. In this example, the categorization of the service providers geographical location is realized by the common taxonomy based on ISO 3166-1999 that is predefined in UDDI repository. The taxonomy of roles is implemented as a 'private' taxonomy described in more detail in Section 4.4.

3.2. Local Workflow Sub-query

BPEL [3] specifies a workflow in terms of tasks (activities in BPEL terminology) representing basic pieces of work to be performed by potentially nested services. The

```
<find_service generic="2.0" xmlns="urn:uddi-org:api_v2">
  <findQualifiers>
    <findQualifier>caseSensitiveMatch</findQualifier>
  </findQualifiers>
  <categoryBag>
    <keyedReference keyName="de:fhg:ipsi:oasys:matchmaking:OTA:Role" keyValue="Agency"
      thModelKey="uuid:A035A07C-F362-44DD-8F95-E2B134BF43B4"/>
    <keyedReference keyName="uddi-org:iso-ch:3166-1999" keyValue="de"
      thModelKey="uuid:4E49A8D6-D5A2-4FC2-93A0-0411D8D19E88"/>
  </categoryBag>
</find_service>
```

Figure 4. UDDI query

control flow of the BPEL process constraints the performance of tasks by choice, sequence, and parallel execution. In addition a BPEL process may also define the data flow of the business process regardless of concrete implementation for the tasks. Based on this understanding, a workflow model includes activities realizing the interaction with partners represented by exchanging messages. So, the set of message sequences supported by a workflow model can be derived. When searching for potential service providers, it is necessary that the exchanged message sequences of the query process are compatible / match to potential trading partners message sequences derived from his local workflow. To be able to check compatibility, the following aspects must be considered:

- Due to the fact that BPEL lacks a formal model [23], a definition of a compatibility operation might be quite vague. It is preferred to have the match operation defined on a solid formal model being able to represent sets of message sequences.
- Testing compatibility is a binary operation, that is all elements contained in a message sequence not related to the opponent must be omitted. Thus, a partner specific view on the own supported message sequence must be calculated.
- The match operation compares the set of message sequences provided by a service provider and contained in the database with the relevant set of message sequences related to the query process.

A detailed description of the following three subsections can be found in [25, 26].

3.2.1 Formal Model

While different formal models representing sets of message sequences exist, in the following, the Workflow Net [22] model is used because it is based on places and transitions like BPEL is, and it provides preliminary results required to express the proposed approach. Other notations like for example, Petri Net [18], statecharts [9], or finite state automata [10] could also have been used.

A Workflow Net (WF-Net) consists of places (circles) representing business tasks and transitions (squares) connecting places representing a message exchange. The transitions are labeled with $s\#r\#msg$ representing sender s and receiver r of the message as well as its message name msg . WF-Nets contain a single final place represented by a circle with a solid line within the graph. A token is depicted as a dot within a circle. A transition is enabled if all input places of a transition contain a token. If a transition is enabled, it might fire, that is removing tokens from incoming places and inserting new tokens to outgoing places of the transition. The current distribution of tokens over the places describes the actual status of the workflow and is named marking.

Figure 5. WF-Net of the accounting department local workflow

The query process of the accounting department as depicted in Figure 3 can be translated into a WF-Net by doing the following mapping of BPEL activities:

- represent *send*, *receive*, *pick*, and *invoke* activities as transitions
- *switch* and *pick* activities represent choices, that is modeled as several transitions each connected with the current place by an input arc
- a *flow* activity represents a parallel execution, that is modeled by replicating the current place in accordance to the number of parallel tasks
- data management operations like *assign* are neglected

The Workflow-Net derived by this transformation is depicted in Figure 5. The process is started by the buyer b sending a $b\#a\#order$ message to the accounting department a , which forwards the order to the logistics department l via a $a\#l\#deliver$ message. The logistics department l confirms this request ($l\#a\#deliver_conf$ message extending the provided information with the planned delivery date and the parcel tracking number) to the accounting department a , which forwards the delivery details of the order ($a\#b\#delivery$ message) to the buyer b . Afterwards, the buyer b is allowed to do parcel tracking with the logistics department l , where the accounting department acts as

a proxy to the buyer b . The corresponding messages are $b\#a\#get_status$ and $a\#l\#get_status$ for requesting the status being answered by $l\#a\#status$ and $a\#b\#status$.

3.2.2 View Generation

The accounting department local workflow depicted in Figure 5 contains messages related to the interaction between buyer and accounting department. Because these messages are irrelevant for the interaction with the logistics department, they must be omitted for calculating the match operation between accounting and logistics workflow. This can be achieved by replacing messages related to the buyer with silent messages as formally specified by projection inheritance on WF-Nets [1]. The transformed WF-Net is then bi-similar (that is a weakened equivalence) to the original WF-Net. A possible view of the accounting department local workflow is depicted as a WF-Net in Figure 6.

Figure 6. view of the logistics department on the accounting department formal local workflow

3.2.3 Match Operation

A match guarantees that the two workflows can interact with each other successfully. A necessary condition to achieve successful interactions is that the workflows provide at least a single common message sequence, i.e. the intersection of the sets of message sequences supported by both workflows is non-empty. The operands of the match operation are the corresponding views of the local workflows.

To be able to illustrate the match operation, Figure 7 depicts an exemplary logistics department local workflow denoted as a WF-Net. Because all messages within this workflow are related to the interaction with the accounting department, the view of the accounting department on this workflow is equivalent to the original workflow depicted in Figure 7. The logistics workflow uses the same messages as the accounting department workflow extended by the $l\#a\#outOfStock$ message representing an out of stock information sent by the logistics department l to the accounting

Figure 7. view of the accounting department on the logistics department formal local workflow

department *a*. The logistics department extends the functionality provided by the accounting department workflow by supporting parcel tracking at any time and providing an option for cancellation of *deliver* messages due to an ordered good being out of stock.

Calculating the match operation of the WF-Nets depicted in Figure 6 and 7 requires that they have a non-empty set of common message sequences. The set of common message sequence is

```
{ a#l#deliver l#a#deliver_conf a#l#terminate
  a#l#deliver l#a#deliver_conf a#l#get_status
    l#a#status a#l#terminate
  : }
```

Thus, the necessary condition is fulfilled. But, to be able to guarantee successful interaction the choices represented in each WF-Net, but not contained in the set of common message sequences must be analyzed. In particular, it must be ensured that all messages that might be sent within the sender's workflow are accepted by the receiving workflow. This sufficient condition is illustrated on behalf of the above example.

While the additional option of the logistics department workflow supporting parcel tracking at any time does not result in message sequences not supported by the accounting department, this does not hold true for the out of stock option. In particular, the logistics department workflow accepting a *a#l#get_status* message at the start state, which is never sent by the accounting workflow does not cause any problem. But, after receiving a *a#l#deliver* message the logistics department might send a *l#a#outOfStock* message, which is not supported by the accounting workflow causing a failing interaction.

Thus, the views of the accounting and the logistics departments local workflows do not guarantee a successful interaction, thus the match operation fails. If the out of stock option in the logistics department workflow is omitted, the

workflow matches the view of the accounting department workflow.

[25] contains a detailed and formal description of a match operation based on extended deterministic finite state automata.

4. Architecture and Implementation

This section describes an architecture and implementation of the IPSI Process Finder **IPSI-PF**. The Process Finder features a number of components all of which are implemented in Java. Figure 9 illustrates the main components of the matchmaking engine.

The input query has three parts: (i) the BPEL part (ii) the WSDL part and the UDDI part. The BPEL part contains process-related descriptions of the query; the WSDL part contains interface descriptions that are referenced by the BPEL part, and the UDDI part provides information that is traditionally provided via the UDDI registry, e.g., business service categories.

4.1. Framework

Processing the query is initiated by submitting a form as exemplary depicted in Figure 8. Triggered by this page the data flow of the architecture depicted in Figure 9 is started, which is realized as a cocoon pipeline. We decided for cocoon, because the data flow between the different processing steps is XML based and thus provides good support by cocoon. In particular, the pipeline realizes the query decomposition as well as the result list merge component depicted in Figure 9. A query decomposition component separates the three parts as follows: BPEL and WSDL parts are sent to the matchmaking engine via a transformation component, **BPEL→formal model**, while the UDDI part is sent to the category matchmaking component. Finally, the merging of the results is also part of this pipeline and is realized by using XSLT.

We decided not to use a more expressive framework like e.g. Struts or a workflow engine due to the fact that the remaining user interactions within the IPSI-PF are much simpler and using the before mentioned concepts seems to be not appropriate.

The implementation is based on the apache project cocoon 2.1.2. The above described pipeline uses *cinclude*, *XSLT* and *session* transformers. A *cinclude* transformer allows to load content of different web resources into a single document. A *XSLT* transformer applies a specified XSLT stylesheet on the intermediate version of the document passed through the pipeline and a *session* transformer grants access to data contained in a HTTP request and stores intermediate results in a session container.

Figure 8. Exemplary Query Form

4.2. BPEL→formal model Component

The role of the **BPEL→formal model** component is to transform the BPEL/WSDL descriptions to a formal model suitable for calculating process matches as described in Section 3.2. The results of the transformation are used as input to the matchmaking engine for process match calculations. This component has been implemented in Java being part of the process matchmaking engine component described next.

Figure 9. Architecture

4.3. Process Matchmaking Engine Component

This component is in charge of performing the comparisons of the query workflow with the workflows stored in the IPSI-PF as introduced in Section 3.2. The formal model

used for calculating business process matches is annotated deterministic finite state automata [25]. Annotated deterministic finite state automata are ordinary deterministic finite state automata [10] with the following additional features:

- the input alphabet consists of triples *sender*, *receiver* and *message name* rather than atomic tokens.
- allows to express optional and mandatory transitions.

The formal approach to matching business processes is presented and discussed in detail in [25].

We decided to use an automaton based approach for the implementation due to missing algorithms for calculating a minimal WF-Net being bi-similar to the projection inheritance of a given WF-Net as described and introduced in Section 3.2. The used automaton approach provides such an operationalization, but suffers from lacking support of modeling concurrent processes. The latter one can be resolved by enumerating all possible sequential combinations introduced by a concurrent partial process.

Each business process in the collection is associated with category data in the UDDI repository via a UUID business service foreign key as shown on Figure 9 used for joining the partial results as described above.

The matchmaking engine is implemented based on the J2EE architecture. The used application server is Sun ONE Application Server 7. The main matchmaking component is an Enterprise Java Bean (EJB) and a web client (not included in the figure) is used for administering the server. Persistency is achieved by container managed persistency relying on a MySQL database system.

4.4. UDDI Category Matching Component

The UDDI engine uses data stored in the UDDI repository to fulfill query requests as introduced in Section 3.1. UDDI related constraints address information stored in a UDDI registry within categorization bags. A categorization is a *name-value* pair assigned to the particular service entity, where the *name* is the name of the used taxonomy and the *value* is a taxonomy value. The UDDI specification provides a number of predefined, common taxonomies, like for example a taxonomy for geographical locations related to ISO 3166-1999. To establish such a taxonomy within UDDI, the specification describes a specific publication process including approval of the registrar to the taxonomy and its impact to the global community using UDDI.

A global taxonomy representing the different roles within the exemplary procurement scenario will not get the approval because it is used only by a specific virtual enterprise. To allow users to categorize services within a local

UDDI registry the UDDI specification provides an alternative approach named "General Keywords taxonomy". Similar to the general taxonomy approach *name-value* pairs are maintained, where the *names* must be prefixed with the organizations own namespace and the supported *values* must be published in the virtual organization internally.

The UDDI implementation used for this prototype is jUDDI [12], which is an open source implementation of the UDDI specification.

4.5. Benchmarking Tool

The benchmarking tool was developed as part of the **IPSI-PF** matchmaking engine project to aid during the design phase of complex business processes. After a complex business process is designed, it is checked for consistency. Consistency checking for complex business processes guarantees that (i) matches do not exist when they are not supposed to, and (ii) matches exist when they are supposed to.

The benchmarking tool is used to run a series of tests on the complex business process, and determine what matches are possible. A reference list can be produced as shown in the figure, that could be useful for validating the consistency of the designed processes providing the possibility the option of evaluating the interactions of the designed complex web services before implementing them.

5. Evaluation

The architecture described above has been implemented and first tests have been performed so far. In the following, a discussion of the applicability of the concept and the implementation is presented as well as initial results of the performance testing.

5.1. Conceptual Discussion

The conceptual approach sketched in this paper can be applied to testing compatibility of bilateral interaction of services. In case of multi-lateral collaborations, the approach is limited to collaborations, where from a global point of view the interactions between the different parties represent a tree structure rather than a graph. The reason for this limitation is that dependencies between different interactions maintained by a single party may result in cyclic dependencies, which could not be recovered by the presented approach. This limitation currently is less restrictive due to fact that current usage of web services is far away of addressing this issue in real world applications.

The main advantage of the concept is that it relies on existing UDDI infrastructure by extending it by querying of workflow models. In particular, the provided match operation increases the precision of the query results and avoids

false matches - that is finding service providers which claim to be compatible without being it - and without having false misses - that is not returning service providers, which are compatible but have not been found by the match operation.

5.2. Implementation Discussion

The current implementation has been realized to evaluate the concept and gain experience in doing querying of workflow models. Thus, it has some limitations, which will be addressed in the next release. In particular, the BPEL→formal model component currently does not support *links* and *join conditions* being part of the BPEL control structure specification, which limits the expressiveness of supported BPEL workflows. An analysis of the overall BPEL expressiveness can be found at [24] being useful to understand the concrete effects of the before mentioned limitation on workflow modeling. While doing experiments with the implementation, we didn't come across a concrete real example, which couldn't have been modeled considering this limitation.

As stated in Section 4.3 the persistence implementation of the process matchmaking engine is based on container managed persistence. This design decision has been made for ease of use and by the knowledge that we nevertheless have to re-implement the persistence model after gaining first experience with the process matchmaking component. We plan that the next release will be based on a bean managed persistence model.

5.3. Initial Performance Testing

The initial performance testing we performed is based on a data set of local workflows in accordance to the IOTP specification. We used IOTP to ensure realistic workflows as well as to achieve an appropriate structural complexity within the used workflows. So, 724 workflows have been created which have been added to the **IPSI-PF** matchmaking engine. To characterize the data set in more detail a description of the structure of the workflow is required. But, this can not be done in a detailed way, thus we give aggregated values of the 724 workflows contained in the storage. After publishing the automata, the storage contains 20180 places, 19370 transitions related to 23 different messages instantiated by 10508 message objects. In addition, it is important to mention, that several of the used automata contain cycles, while others are tree-structured.

The minimum processing time is 13 seconds, the maximum nearly 3 minutes, and the mean processing time is about 1 minute. These measures are done on a laptop with 512 MB RAM and all required software running on it. Nevertheless, the performance is not sufficient for online service discovery and requires an investigation of indexing mecha-

nisms. We started this task already by doing a survey of existing indexing mechanisms. The presented results are first initial testing. This work will be continued by performance tests taking into consideration the effects of workflow structures on these measures.

6. Related Work

Service discovery is a popular research topic addressed by different communities focusing on different aspects. All proposals have in common to extend the limited functionality as provided by UDDI.

Overhage and Thomas [16] propose a comprehensive framework for web services discovery to supersede the current UDDI framework. The framework, named WS-specification, categorizes web services into white, yellow, blue and green pages. The terms, white, yellow, blue and green pages are adopted from UDDI, but extended and sometimes redefined in WS-specification. White pages for example contain information about general and technological (architecture specification, performance, security) information about services. Yellow pages contain classification information (like in UDDI), blue pages contain conceptual information (process-related, semantics) and green pages contain interface information. Among the observations made in [16] is the deficiency of UDDI specification for not describing process-related information of services. Process-related information is thus provided for in the blue pages of WS-specification. They propose BPEL (used in this paper) as a possible formal definition language for business process descriptions. We are however not aware of any implementation of this specification so far and no clue about querying web services based on process information is provided.

Logic based approaches addressing service discovery are Web Service Request Language (WSRL) and Product Lifecycle Management PLM_{flow} . WSRL [2] addresses planning of an orchestration and composition of services to fulfill user requirements. While WSRL performs service discovery on behalf of temporal and linear constraints, PLM_{flow} [28] is based on rule inferencing using the specified business rules rather than a fixed workflow. Thus, PLM_{flow} is characterized as a rule-based non-deterministic workflow engine aiming to establish cooperation based on local decidability of the trading partners of their involvement. These approaches are based on the fact that the local workflow model/rules are provided to the trading partners and do not consider the need of hiding business critical information as is the case with abstract business processes which are addressed in this paper.

In [15], two services are regarded as compatible if every possible trace in one service has got a compatible one in the second one. This approach is similar to that used in

this paper, but unfortunately, the description how to do the compatibility check of the traces is not given.

Matchmaking of services is also being addressed in the semantic web community [5, 14]. The used approach is to add semantic annotations to service descriptions. Services are searched on the basis of what they can do, i.e., what features and capabilities they have. Standards such as DAML-S [4] are used to describe such services. This is the approach taken in [13], where the UDDI search functionality is extended by adding semantic-based search functionality, to search for services, in addition to what UDDI provides. The concept of semantic annotation of web service descriptions for matchmaking is also used in [17, 20, 8]. The main drawback of semantic annotation is the necessity of a common ontology used for annotating and querying services. Unfortunately, no such ontology currently is in place.

AGFlow is an agent-based cross-enterprise workflow management system for discovering services. Service discovery is based on their location and predefined properties [27]. The approach does not describe how process-related information can be encoded into the agent, and used as a possible criteria for service discovery.

In [7, 6] business process modeling approaches are described to integrate processes within and outside the enterprise. The approach presented in [6] is based on extending workflow systems with the ability to register to events, and to receive events at specific points within the workflow. In [7], a comprehensive framework is given how multiple virtual enterprises can integrate their processes to work together. The proposed approaches do not however describe how to check for compatible business processes in a dynamic environment as is done in this paper.

In [19], UDDIe is presented as an extension to UDDI, and its implementation is provided as open source at [21]. UDDIe extends UDDI in three main ways: (i) service leasing support, (ii) introducing properties for describing services and the ability to search on these properties and (iii) extending the UDDI *find* API with the ability to support numeric and logical *AND/OR* operators. This extension provides a more expressive way to describe and query services than originally provided by UDDI, but does not consider the business process aspect of the services as is done in this paper.

7. Summary and Future Work

The paper presented the concept, architecture and implementation behind the **IPSI-PF** process matchmaking engine, including an evaluation. The **IPSI-PF** process matchmaking engine readily fits to the existing web service framework, and it complements UDDI service discovery by extending it with the ability to search for compatible business partners through comparison of their business processes.

Future releases of this engine will include an index to efficiently find compatible partners where a large amount of business process data is involved. Finding a suitable index for business process matching is complicated by the fact that, unlike flat key search this type of data does require comparison operations interpreting a structure rather than allowing to compare structures.

References

- [1] W. Aalst. Inheritance of Interorganizational Workflows to Enable Business-to-Business E-commerce. *Electronic Commerce Research*, 2(3):195–231, 2002.
- [2] M. Aiello, M. Papazoglou, J. Yang, M. Pistore, M. Carman, L. Serafini, and P. Traverso. A request language for web services based on planning and constraint satisfaction. In *Proc. of 3rd Int. Workshop, Technologies for E-Services (TES)*, LNCS 2444, pages 76–85. Springer, 2002.
- [3] T. Andrews, F. Curbera, H. Dholakia, Y. Golland, J. Klein, F. Leymann, K. Liu, D. Roller, D. Smith, S. Thatte, I. Trickovic, and S. Weerawarana. Business process execution language for web services. version 1.1. <http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnbiz2k2/html/BPEL1-1.asp>, May 2003.
- [4] D.-S. C. A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. McDermott, D. Martin, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. Daml-s: Web service description for the semantic web, 2002. <http://citeseer.nj.nec.com/ankolekar02damls.html>.
- [5] T. Berbers-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific America*, 284(5):34–43, 2001.
- [6] F. Casati and A. Disenza. Modeling and managing interactions among business processes. *Journal of Systems Integration*, 10(2):145–168, Apr. 2001. Special Issue: Coordination as a Paradigm for Systems Integration.
- [7] D. Georgakopoulos, H. Schuster, A. Cichocki, and D. Baker. Managing process and service fusion in virtual enterprises. *Information Systems*, 24(6):429–456, September 1999.
- [8] J. Gonzalez-Castillo, D. Trastour, and C. Bartolini. Description logic for matchmaking of services. Technical Report HPL-2001-265, Hewlett-Packard, October 2001.
- [9] D. Harel. Statecharts: A visual formalism for complex systems. *Science of Computer Programming*, 8(3):231–274, June 1987.
- [10] J. E. Hopcroft, R. Motwani, and J. D. Ullman. *Introduction to Automata Theory, Languages, and Computation*. Addison Wesley, 2001.
- [11] IBM, Microsoft, HP, Oracle, Intel, and SAP. Universal description, discovery and integration, July 2002. <http://www.uddi.org/>.
- [12] juddi. <http://ws.apache.org/juddi>.
- [13] T. Kawamura, J. D. Blasio, T. Hasegawa, M. Paolucci, and K. Sycara. Preliminary report of public experiment of semantic service matchmaker with uddi business registry. In M. E. Orlowska, S. Weerawarana, M. P. Papazoglou, and J. Yang, editors, *Service oriented computing - ICSOC 2003 LNCS 2910*, pages 208–224. Springer, 2003.
- [14] S. McIlraith, T. Son, and H. Zeng. Semantic web services. *IEEE Intelligent Systems (Special Issue on the Semantic Web)*, April 2001.
- [15] M. Mecella, B. Pernici, and P. Craca. Compatibility of e-services in a cooperative multi-platform environment. In F. Casati, D. Georgakopoulos, and M. Shan, editors, *TES 2001 LNCS 2193*, pages 44–57. Springer, 2001. Lecture notes in Computer Science 2193.
- [16] S. Overhage and P. Thomas. Ws-specification: Specifying web services using uddi improvements. In A. Chaudhri, M. Jeckle, E. Rahm, and R. Unland, editors, *Web, Web-Services, and Database Systems: NODE 2002, Web- and Database-Related Workshops, Erfurt, Germany*, volume Volume 2593 / 2003, pages 100 – 119. Springer-Verlag Heidelberg, 2002. Lecture Notes in Computer Science.
- [17] M. Paolucci, T. Kawamura, T. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the First International Semantic Web Conference on The Semantic Web*, pages 333–347. Springer-Verlag, 2002.
- [18] J. L. Peterson. *Petri Net Theory and the Modeling of Systems*. Prentice-Hall, 1981.
- [19] A. ShaikAli, O. F. Rana, R. Al-Ali, and D. W. Walker. Uddie: An extended registry for web services. In *Proceedings of the 2003 Symposium on Applications and the Internet Workshops (SAINT-w03)*. IEEE Computer Society, 2003.
- [20] K. P. Sycara, M. Klusch, S. Widoff, and J. Lu. Dynamic service matchmaking among agents in open information environments. *SIGMOD Record*, 28(1):47–53, 1999.
- [21] uddie. Uddie homepage, 2003.
- [22] W. van der Aalst and K. van Hee. *Workflow Management - Models, Methods, and Systems*. MIT Press, 2002.
- [23] W. van der Aalst. Dont go with the flow: Web services composition standards exposed. *IEEE Intelligent Systems*, pages 72–76, Jan/Feb 2003.
- [24] P. Wohed, W. M. van der Aalst, M. Dumas, and A. H. Hofstede. Pattern based analysis of bpel4ws, 2002.
- [25] A. Wombacher, P. Fankhauser, B. Mahleko, and E. Neuhold. Matchmaking for business processes based on choreographies. In *Proc. of International Conference on e-Technology, e-Commerce and e-Service (EEE-04)*. IEEE Computer Society Press, 2004.
- [26] A. Wombacher, P. Fankhauser, and E. Neuhold. Transforming bpel into annotated deterministic finite state automata enabling process annotated service discovery. In *Proc. of IC Web Services*, 2004.
- [27] L. Zeng, B. Benatallah, P. Nguyen, and A. H. H. Ngu. Agflow: Agent-based cross-enterprise workflow management system. In P. M. G. Apers, P. Atzeni, S. Ceri, S. Paraboschi, K. Ramamohanarao, and R. T. Snodgrass, editors, *Proc. of 27 international conference. on VLDB*, pages 697–699, 2001.
- [28] L. Zeng, D. Flaxer, H. Chang, and J.-J. Jeng. PLMflow - dynamic business process composition and execution by rule inference. In *Proc. of 3rd Int. Workshop, Technologies for E-Services (TES)*, LNCS 2444, pages 141–150. Springer, 2002.