Toward a Semantic Web service discovery and dynamic orchestration based on the formal specification of functional domain knowledge

Pierre Châtel
LIP6/Thales Communications
1 à 5, avenue Carnot
91883 Massy Cedex - France
Tel: +33(0)1 69 753 057 – e-mail: pierre.chatel@fr.thalesgroup.com

Abstract: This paper presents the work that has been done at Thales Communications France regarding the specification and implementation of a semantic Web Service discovery and orchestration platform as part of the S4ALL ITEA program [13]. The central objective is to increase systems' interoperability and adaptability by adding semantic annotations on service declarations and business processes, in order to capitalize on the information contained in ontologies. The semantic platform was developed using common (semantic) web languages and technologies like the Semantic Annotation for WSDL Language (SAWSDL), BPEL, OWL and the UDDI service registry. The following twofold approach has been undertaken: firstly, specifying a SAWSDL to UDDI structural mapping. Its implementation allows us to publish semantically annotated declarations into a well-known service registry and make discovery queries regarding semantic concepts extracted from ontologies. Secondly, implementing a semantically aware business process execution engine able to conduct reasoning on processes where service requirements are expressed using ontological concepts. It also supports a basic form of data adaptation.

Key words: Web service composition and orchestration, interoperability, business process, dynamic configuration, adaptive systems, web-services, ontologies

1. INTRODUCTION

The work presented in this paper is an attempt to solve some of the inherent problems in highly dynamic and changing distributed environments where Web services can appear and disappear during runtime: in such environments, an high level of interoperability between service providers and consumers is required and very few binding choices can be made at design time because services are not predefined and mostly discovered on the fly.

Following current trends, these kinds of characteristics could be found in most of Thales' future military and civilian activities. Take for example the need, on battlefield, to deploy an information and command system shared between several allies in an international coalition (e.g. NATO). In this type of command system, business process correspond to usual military procedures and can be designed in advance, but process engines will need to dynamically discover the various entities (e.g. infantry, artillery) and to adapt themselves according to their eventual unavailability. There is also the necessity, in civilian crisis management systems, to discover and coordinate protagonists in a timely fashion and enable particular rescue scenarios according to the available networked equipments.

We choose to particularly study the case where a common understanding of the domain is shared between the protagonists but where discrepancies may exist at the semantic and syntactic level between service consumers' requirements and providers' declarations. Also, we focus on the highly dynamic Web service orchestration scenario where Web services are progressively bound to business processes at runtime and on-demand.

The paper is organized as follow. Section 2 explains the technological choices that were made to meet our objectives and allow the implementation of a supporting platform. Section 3 gives details on our methodology to store semantic Web services into UDDI. Then, in sections 4 and 5, we focus on the reasoning abilities over ontologies of our platform and on the implementation of a semantically-aware BPEL execution engine around this capacity. Before concluding we will forecast, in section 6, upcoming works on non-functional constraint handling that could be added to this realization.

2. TECHNOLOGICAL CHOICES

Shared knowledge on a given domain will be specified using ontologies. In these user-specified knowledge bases, concepts and relations are defined to represent noteworthy entities. We have chosen the OWL-DL ontology language [8] for its formal ground in Description Logics, numerous reasoners [1,4,10], extensibility and widespread adoption in semantic Web communities. This should allow us to benefit from existing OWL-DL domain ontologies.

In order to allow more powerful matches between service consumers' requirements and providers' declarations than what is commonly possible at the syntactic level, we provide a methodology to extend pre-existing service descriptions with semantic annotations. As such, this platform is, to our knowledge, one of the first to present a practical application of the SAWSDL W3C Candidate Recommendation [5]. This WSDL 2.0 extension enables a service provider to write semantically enhanced service declarations that adds to the usual syntactic level (e.g. endpoints, protocols, parameters types). Compared to complete overhauls of service descriptions formalisms like OWL-S [7], the SAWSDL specification does not require service providers to learn a new language and can be used to semantically extend previously defined descriptions.

Since Web services are dynamically discovered at runtime in our platform, multiple discovery patterns could have been used ranging from services registries to totally decentralized protocols. We have chosen an UDDI Web service registry [12], jUDDI[1], for its low network traffic overhead, ease of deployment and wide adoption. In our implementation, the UDDI registry will be used to store and query the semantic aspects of the SAWSDL declarations.

In order to orchestrate Web service functional capabilities into useful business processes, we take the common industrial approach of using the BPEL specification [2]. A BPEL process allows the specification of macro-services that will be composed at runtime from atomic services advertised in the registry. The ActiveBPEL[2] execution engine has been chosen for its extensibility and user-friendly BPEL editor. These BPEL processes are the exclusive service consumers in our platform and benefit from the same semantic enhancements as the service declarations.

3. STORING SEMANTIC WEB SERVICE DESCRIPTIONS INTO A SERVICE REGISTRY

In an environment where Web services can appear and disappear at anytime, services need to advertise themselves to their potential consumers. To do so, as mentioned earlier, a central registry will be used and it is the responsibility of services providers to proactively register their services into the registry. On the other hand, service consumers (business processes in our case) need to be able to dynamically discover and select services according to their needs.

The distinctive feature of the semantic Web service approach is to allow the storage of semantic information in the registry in order for service consumers to select services based on this implementation-independent information. It should also be understood that all protagonists need to share a common understanding of the domain: this knowledge will be conveyed by OWL-DL ontologies.

Since an UDDI service registry does not allow, as-is, to store any semantic information related to service declarations, we defined an SAWSDL to UDDI mapping and based a service publication and querying API named LUCAS (Layer for UDDI Compatibility with Annotated Semantics) around it. This particular mapping stores the semantic attributes of a service within *category bags* of related *tModels*. Both *category bag* and *tModel* are UDDI-specific constructs that can be queried using the standard UDDI API.

The primary goals that were attained with this mapping are enabling the automatic registration of SAWSDL definitions in UDDI and the execution of precise and flexible UDDI queries based on specific semantic metadata. We were also looking for the optimization of UDDI's query processing performance and to support any logical and physical structure of SAWSDL description while maintaining compatibility with the previous WSDL 2.0 to UDDI mapping.

In its current state, LUCAS provides a query API for searching services in a registry according to their methods' semantic goals, inputs or outputs, independently of syntactic or data-types concerns (cf. Figure 1).

[1] http://ws.apache.org/juddi/
[2] http://www.active-endpoints.com

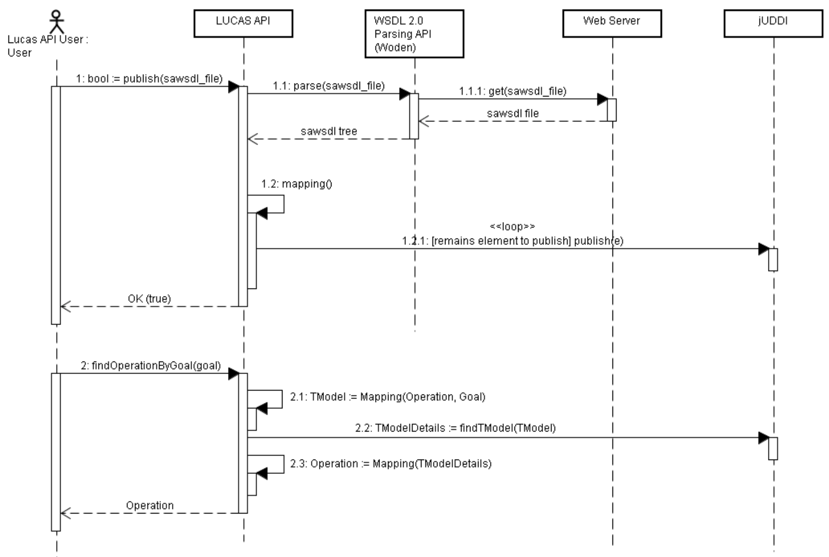

Lucas API User : User
LUCAS API
WSDL 2.0 Parsing API (Woden)
Web Server
jUDDI
1: bool := publish(sawsdl_file)
1.1: parse(sawsdl_file)
1.1.1: get(sawsdl_file)
sawsdl file
sawsdl tree
1.2: mapping()
<<loop>>
1.2.1: [remains element to publish] publish(e)
OK (true)
2: findOperationByGoal(goal)
2.1: TModel := Mapping(Operation, Goal)
2.2: TModelDetails := findTModel(TModel)
2.3: Operation := Mapping(TModelDetails)
Operation

Figure 1 - SAWSDL publishing and registry querying with LUCAS

4. SEMANTIC EQUIVALENCE

In our implementation, a service request and a service offer will match if they can be considered semantically equivalent, even with slight discrepancies between the two. According to the desired results, there are various ways to consider semantic equivalence. We designed LUCAS so that it can be expanded to accept different definitions of equivalence but we also provide a base equivalence algorithm.

This algorithm has a low computing overhead and bases equivalence on the 'is-a' (rdfs:subClassOf) and 'is-equivalent-to' (owl:EquivalentClass) OWL-DL relations indicated between classes in single inheritance ontologies. It does not consider service declarations as a whole but focuses on operations.

An operation is considered as a vector of semantic information. Let V_{Req} be an operation requirement and V_{Dec} be an operation declaration where goal, inputs and outputs are ontology concepts, we have:

$V_{Req} = <goal_{Req}, inputs_{Req}, output_{Req}>$

$V_{Dec} = <goal_{Dec}, inputs_{Dec}, output_{Dec}>$

V_{Req} and V_{Dec} are considered semantically equivalent if $goal_{Req}$ is a subconcept of, or is equivalent to $goal_{Dec}$; if each element of $inputs_{Req}$ is a subconcept of, or is equivalent to an element in $inputs_{Dec}$; and if $output_{Req}$ is a super-concept of, or is equivalent to $output_{Dec}$.

To illustrate this basic reasoning implementation, consider the following example where methods are described using ontology classes (cf. Figure 2). Only one available method is matched because it's the only one that validates the previous definition.

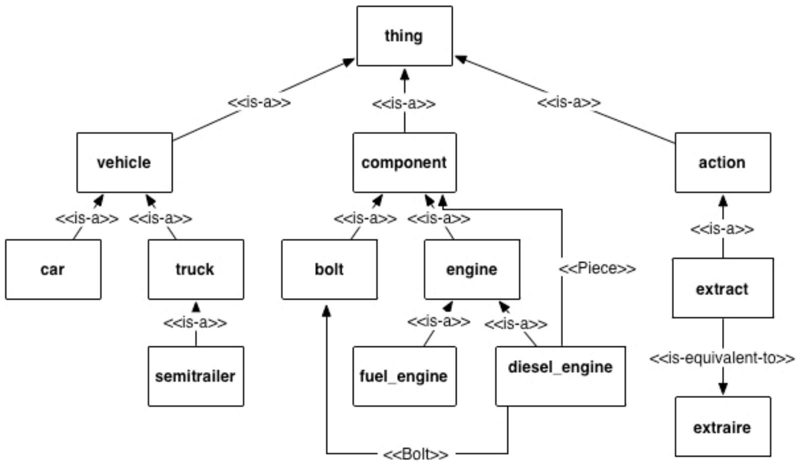


Figure 2 - Exemple ontology

Required operation	Available operations in Web services	Matching	Comment
extract(truck):engine	extraire (vehicle):diesel_engine	OK	
	extract (**semitrailer**):engine	NOK	The input concept is not in a contravariance relationship with the requested input. It is too narrow.
	extract (vehicle):**component**	NOK	The output concept is not in a covariance relationship with the requested output. It is too wide.

It is important to note that other reasoning techniques could be developed for LUCAS in the future. For instance, using the *subsumption* mechanism offered by a Description Logic reasoner to infer implicit 'is-a' and 'is-equivalent-to' relations between OWL concepts in an ontology and, as such, increase the scope of our semantic matchmaking algorithm (which currently only uses explicit relations between concepts).

5. SEMANTICALLY AWARE BPEL EXECUTION ENGINE

In our semantic SOA platform, two distinct roles are defined:

- **Service provider**: which provide and publish various basic services. We explained in the previous sections how services providers could register their semantic service offers as SAWSDL instances into a central UDDI registry (cf. 3) and how a service request could be matched against multiple service declarations (cf. 4).
- **Service consumer**: on the other hand, service consumers need to organize the basic services available on the network into more elaborate and useful business processes. As such, services can be seen as the primary building blocks of bigger structures: BPEL processes.

But, when designing a BPEL process, the required web services may not already be available. And even if they are, they may not be available at runtime. Indeed, we have to keep in mind that designing a BPEL process is an early step in the lifecycle of a semantic SOA application, whereas web services can appear and disappear at any time.

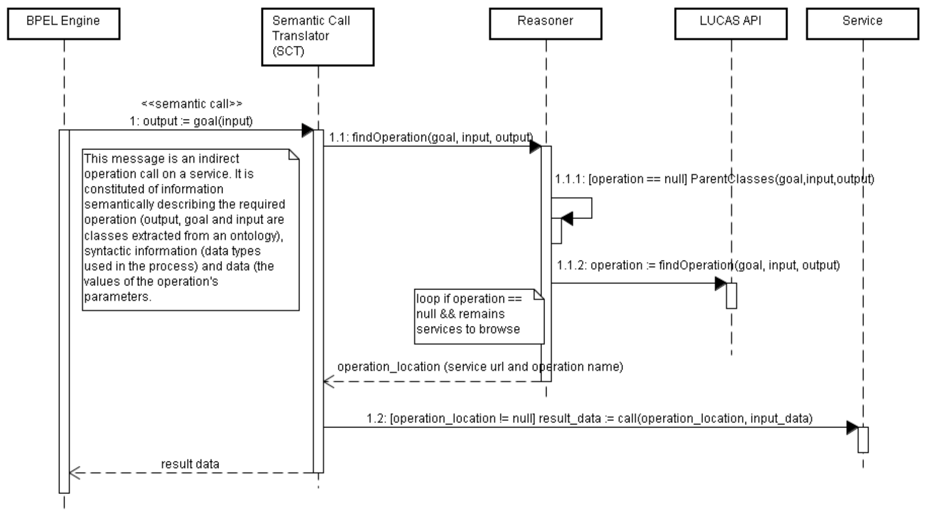
From these basic facts, stems the need to make a strict distinction between consumers' service requirements (statically and semantically defined in the BPEL process) and the web services that will fulfill them (dynamically selected at runtime).

5.1. ActiveBPEL engine modification: the Semantic Call Translator

As such, the final component of our platform is a semantically aware BPEL engine that uses the semantic information contained in the processes in order to filter the available services at runtime according to its needs.

The ActiveBPEL engine that we are using does not provide any semantic reasoning facilities. In order to minimize modifications to the engine and maintain compatibility with standard BPEL processes, we only have slightly modified its source code to forward all the semantic reasoning responsibility to an other component of our architecture, the 'Semantic Call Translator' (SCT).

The SCT is implemented externally from the BPEL engine and act as a proxy by intercepting its semantic Web service operation calls (cf. Figure 3). It is its duty to select the appropriate service on the fly by conducting semantic reasoning between consumer's requirements and services' advertisements (the algorithm presented in section 4 has been implemented and used to this end). In fact, the whole semantic call remains transparent to the BPEL engine itself.


BPEL Engine
Semantic Call Translator (SCT)
Reasoner
LUCAS API
Service
<<semantic call>>
1: output := goal(input)
1.1: findOperation(goal, input, output)
1.1.1: [operation == null] ParentClasses(goal,input,output)
1.1.2: operation := findOperation(goal, input, output)
operation_location (service url and operation name)
1.2: [operation_location != null] result_data := call(operation_location, input_data)
result data

This message is an indirect operation call on a service. It is constituted of information semantically describing the required operation (output, goal and input are classes extracted from an ontology), syntactic information (data types used in the process) and data (the values of the operation's parameters.

loop if operation == null && remains services to browse

Figure 3 - End to end semantic call handling

In order to select the appropriate service at runtime, the SCT uses the UDDI querying subsection of the LUCAS API (cf. 3, 4). If no exact semantic match can be found, LUCAS will automatically make multiple queries to the UDDI registry until a match is found or no valid service can be selected.

5.2. Data adaptation

Another key aspect of service orchestration is to allow transparent data flow between services consumers and producers. We hereby present an original solution based on the formal specification of functional domain knowledge: the domain ontologies.

Considering the fact that web services are dynamically selected during orchestration and that our previous definition of equivalence between a request and an offer is based on semantic information, the need to handle the inferred data adaptation at the syntactic level by the SCT becomes obvious.

Indeed, considering a requesting BPEL process, the matched service offer does not necessarily possesses the same data-types for its input and output as the ones defined in the process, hence a basic data adaptation methodology should at least be transparently implemented by the SCT.

The SAWSDL specification does anticipate this particular need since it offers two special annotations on the subject: namely *liftingSchemaMapping* and *loweringSchemaMapping*. These annotations, when applied properly to XML Schema data type definitions, allow us to make bi-directional data conversion between service providers and consumers.

This conversion relies on the fact that the matched data types are semantically equivalent: we can assume that if two distinct XML Schema types are annotated by the same ontological concept then they must designate a common notion in the studied domain and, as such, convey the same information. In this case, the domain ontology can be used as a pivot data model between the service requester and provider:

- The *liftingSchemaMapping* annotation will point to an XSLT stylesheet specifying the transformation of data instances from the peer's own types (requester or provider) to ontological concepts instances.
- The *loweringSchemaMapping* annotation will point to an XSLT stylesheet specifying the transformation of ontological concepts instances to data instances from the peer's own types (requester or provider).

In the following example (cf. Figure 4), the service requester (a BPEL process) and the service provider (a Web Service) use different data types to store the same kind of information (an order request constituted of multiple items). The lifting and lowering annotations allow the SCT to convert order request data back and forth between both sides and, as such, allowing orchestration.

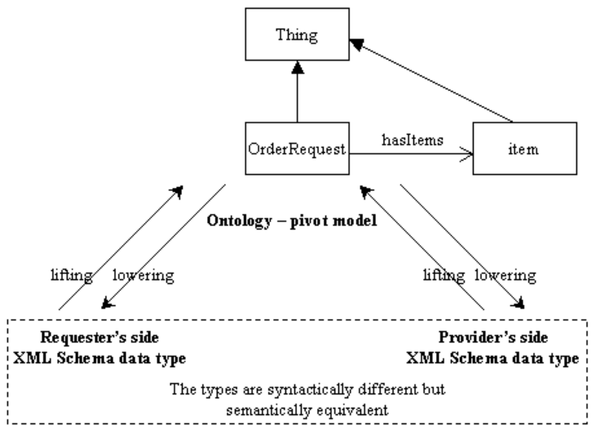


Figure 4 - Ontology as a pivot data model

It is to be noted that specifying the lifting and lowering schemaMapping is the responsibility of both service providers and requesters since it needs to be done before service or process deployment. So this is not a fully automated paradigm and still necessitates human intervention, this is why we believe it can be improved.

6. ONGOING WORK: NON-FUNCTIONAL CONSTRAINTS HANDLING

The work that was presented in this paper needs to be extended outside purely functional concerns in order to increase its range of usability. Indeed, due to the highly dynamic nature of service orchestration, **we need to build Quality of Service (QoS) awareness into the platform**. This subject is currently being investigated in a joint PhD thesis between Thales Communications France and the LiP6 computer-science laboratory[3].

We are trying to establish a dynamic service orchestration based not only on a formal specification of functional domain knowledge but also on **non-functional constraints and instantaneous QoS measurements on service providers.**

In this generalized and extensible SOA framework, several core features collaborate in order to handle the end-to-end execution of a BPEL Process. These features are:

- Enhanced BPEL process execution (BPEL + functional and non-functional constraints).
- Functional filtering of service offers based on a service request and available ontologies.
- Non-functional filtering of service offers based on the process's Quality of Service requirements.
- Dynamic non-functional selection of the best available service based on business preferences and instantaneous QoS measurements on service providers.

7. RELATED WORK

Most of the work currently available on semantic service discovery and orchestration only cover a subset of the features we want to implement within this framework. They focus, for the most part, on the semantic matching between requests and offers but don't take into account non-direct semantic equivalence between offers and requests (cf. 4), data adaptation between service requesters and providers (cf. 5.2) or non-functional constraint handling (cf. 6), despite these steps appear mandatory throughout a service call.

Other companies or laboratories have already separately investigated some of the concepts that are encompassed by our SOA framework. But the originality of this realization is to join them in a pragmatic fashion inside a specific architecture and working prototype in order to support end-to-end dynamic semantic service calling.

The following works use functional ontologies to describe the various "capabilities" of Web Services but present different approaches for matchmaking.

Li & Horrocks present in [6] a prototype using a DAML-S ontology and a description logic reasoner to compute the subsumption relationship between a service offer and request, but they don't provide any solution for the data adaptation problem that arises after matchmaking.

In other cases, services are described by the state transformation they generate. The comparison between offers and requests is then grounded on the semantic concepts used to describe services' inputs and outputs. It is this

[3] http://www.lip6.fr/en/index.html

particular approach that was generalized in our own framework. The subsumption relationship is also put to good use, but not at the same abstraction level than in [6] (it is not computed directly between an offer and a request).

In [9], DAML-S was chosen to describe service offers. Paolucci et al. tackle the matchmaking problematic between offers and requests from the point of view of their semantic description but fail to give much detail about their implementation. Also, as in our implementation, they keep a UDDI registry for service publication and discovery.

In [3], Di Noia et al. have implemented a matchmaking process based on the description logic subsumption between concepts. They specifically detail the corresponding algorithm and distinguish three matchmaking degrees: total, potential and partial.

Sycara et al. present in [11] the LARKS language for specifying service offers and requests and a matchmaking process that integrate both semantic and syntactic aspects. They use the ITL ontology language and subsumption relationship but are not specifically related to service oriented architectures.

8. CONCLUSION

In this paper we have shown how a semantic Web service platform can be implemented from independent and interchangeable modules (publisher, inquirer, reasoner, semantic call translator, BPEL engine). Based on widely available open-source technologies from the Semantic Web and Web service communities, this platform can be deployed across Thales's range of competence and in inter-corporation networks (since it is not linked to any specific profession).

For Thales, this platform is the first step toward a useful and practical semantic Web service discovery and dynamic orchestration service that could be reused indiscriminately in its future military and civilian activities. Ongoing work in the domain of non-functional constraints handling should foster its adoption.

Although basic, solutions for data adaptation and ontology reasoning were presented in this paper and implemented in our semantic SOA framework (LUCAS + SCT + BPEL Engine). Some problems remain and are currently investigated: since our framework has been designed with extensibility in mind, this should allow us to 'plug' new data adaptation and semantic reasoning techniques into it.

9. ACKNOWLEDGEMENT

Hugues Vincent – Thales Communications France
hugues.vincent@fr.thalesgroup.com
1 à 5 avenue Carnot
91883 Massy Cedex - France

Jacques Malenfant – Laboratoire d'Informatique de Paris VI (LIP6)
Jacques.Malenfant@lip6.fr
Université Pierre et Marie Curie
104, avenue du Président Kennedy
F-75016 Paris – France

10. REFERENCES

[1] Bechhofer S. The Hoolet OWL-DL reasoner, http://owl.man.ac.uk/hoolet/

[2] Bolie, J.; Cardella, M.; Blanvalet, S.; Juric, M.; Carey, S.; Chandran, P.; Coene, Y.; Geminiuc, K.; Zirn, M. & Gaur, H. BPEL Cookbook: Best Practices for SOA-based integration and composite applications development Packt Publishing, 2006

[3] Di Noia, T.; Di Sciascio, E.; Donini, F. & Mongiello, M. *Semantic matchmaking in a P-2-P electronic marketplace*, Proc. Symposium on Applied Computing (SAC'03), 2003, 582-586

[4] Jang, M. & Sohn, J. Bossam: An Extended Rule Engine for OWL Inferencing, Workshop on Rules and Rule Markup Languages for the Semantic Web at the 3 rdInternational Semantic Web Conference (LNCS 3323), Springer, 2004, 128-138

[5] Lausen, H. & Innsbruck, D. Semantic Annotations for WSDL and XML Schema 2007

[6] Li, L. & Horrocks, I. A Software Framework for Matchmaking Based on Semantic Web, Technology International Journal of Electronic Commerce, ME Sharpe, 2004, 8, 39-60

[7] Martin, D.; Burstein, M.; Hobbs, J.; Lassila, O.; McDermott, D.; McIlraith, S.; Narayanan, S.; Paolucci, M.; Parsia, B.; Payne, T. & others OWL-S: Semantic Markup for Web Services 2004

[8] McGuinness, van Harmelen et al: OWL Web Ontology Language Overview, W3C Recommendation, 2004

[9] Paolucci, M.; Kawamura, T.; Payne, T. & Sycara, K. Semantic Matching of Web Services Capabilities, Proceedings of the 1st International Semantic Web Conference (ISWC), Springer, 2002, 348

[10] Parsia, Sirin, Pellet: An OWL DL Reasoner, Proceedings of the International Workshop on Description …, 2004

[11] Sycara, K.; Widoff, S.; Klusch, M. & Lu, J. Larks, Dynamic Matchmaking Among Heterogeneous Software Agents in Cyberspace Autonomous Agents and Multi-Agent Systems, Springer, 2002, 5, 173-203

[12] Walsh, A. Uddi, Soap, and Wsdl: The Web Services Specification Reference Book Prentice Hall Professional Technical Reference, 2002

[13] ITEA 2 Blue Book, European leadership in Software intensive Systems and Services, http://www.itea-office.org/publications