The Adaptation of Agent Configurations using Web Services as Components

D. Richards[1], S. van Splunter[2], F.M.T. Brazier[2], M. Sabou[2]

[1] Department of Computing
Macquarie University
Sydney, Australia
richards@ics.mq.edu.au

[2] Department of Computer Science
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands
< marta, sander, frances>@cs.vu.nl

Abstract*: To support dynamic and reactive behaviour we have developed an Agent Factory which (re-)structures an agent configuration using existing components. In this paper we present our current work which uses Web services as the components. Our initial work has focused on configuring Web services to create a design artifact. This is achieved by reasoning about the requirements and the semantic descriptions of Web services in the DAML-S Web service description language and ontologies. We provide an example of the design process and our findings regarding DAML-S.*

Keywords: Applications, Intelligent agents, Ontology, Semantic Web, Web services

1. Introduction

The ability of agents to adapt according to changes in system requirements and the environment is important to enable dynamic and reactive behaviour. Following the use of compositionality in the major software engineering paradigms and based on the Factory design pattern [8], we have developed an Agent Factory (AF) architecture [1]. The approach is based on the use of components, the generic agent model and the DESIRE formal knowledge-level modeling and specification framework for multi-agent systems [3]. Our agent (re-)structuring approach [13] allows an agent to automatically adapt by reusing existing components. Our approach is a combination of process-oriented and object-oriented approaches by treating processes as the 'active' parts of our agent, which are our agent components, and classes as the 'passive' part of our agent, which are the data types used in the agent components.

Adaptation requires the identification of appropriate components. Determining what is 'appropriate' involves understanding of the requirements, the development of a design to meet those requirements, location of possible components and the ability to reason about those components. Reusable components must therefore be described syntactically and semantically to determine if a suitable component has been found and what changes, if any, are needed. There are a number of possible alternatives to using reusable components ranging from one extreme of creating your own closed library of components with well constrained specifications, languages, etc, or the other extreme of finding components "out there" which will require sophisticated matching and adaptation techniques. We have sought a middle position on this continuum. Our previous work has proposed an open architecture but our implementations have been restricted to the use of building blocks that we have developed our selves. Furthermore, the Agent Factory has been developed on the basis of a number of assumptions:

1. agents have a compositional structure
2. reusable parts can be identified
3. two levels of descriptions are used: conceptual and implementation
4. properties and knowledge of properties are available
5. no commitments are made to specific languages and/or ontologies.

Our current work is reviewing those assumptions through the application and extension of our approach by using Web services as components. Web services meet our first assumption as they exhibit modular behaviour [5]. The study described in this paper is particularly focused on addressing the second and fourth assumptions. Many agent approaches are based on similar assumptions and thus our work is of benefit to them. Further we believe that our architecture can be used to create composite Web services and address some of the issues facing the Web services and Semantic Web communities.

Additionally, the study reported in this paper seeks to take some of the key research efforts and defacto standards that are emerging for automated Web service usage and investigate their strengths and deficiencies. In the next section we provide some background to this project through review of a number of other adaptive agent approaches and current approaches to dynamic use of Web services. In section 3, we introduce the Agent Factory including a description of components, our architecture, the general agent model and assumptions. The fourth section describes the study we conducted. The final section gives our conclusions, future work and summary.

2. Background

To provide some background to our work we present an introduction to Web services and their semantic description in sections 2.1 and 2.2. Section 2.3 considers agent-based Web service research.

2.1 Web services

While a number of definitions of a Web service exist, the definition that most fits with our intended use of WSs as

components in the Agent Factory is given by the Stencil[i] group who define a WS as "loosely coupled, reusable software components that semantically encapsulate discrete functionality and are distributed and programmatically accessible over standard internet protocols". The three definitions offered differ in their emphasis on technology, business and software engineering but all encapsulate the self-contained, modular, composable and distributed nature of WS. These four characteristics of WS are well supported by a layered-architecture where the base is a well-established transport layer. In each layer we give an example of a major standard. In italics we position the work reported in this paper. The next layer up uses the Simple Object Access Protocol (SOAP) which is an XML-based communication protocol for exchanging data in

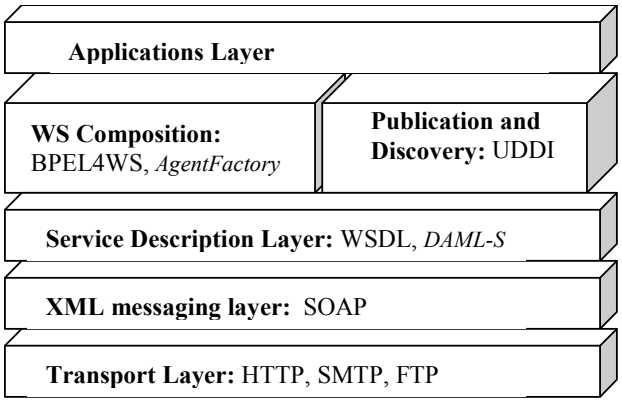

Figure 1: WS architecture (adapted from [14])

decentralized and distributed environments via typed message exchange and remote calls. The service description layer includes the XML-based Web Service Description Language (WSDL). The next layer is split into two main types of WS technologies: ones that support single service advertising and discovery and ones that support service composition. For service registration and discovery there is the Universal Description, Discovery and Integration (UDDI) (by IBM, Microsoft and Ariba) standard service repository. To provide some very basic semantics (such as identification via a product classification code) one or more tModel descriptions may be attached to a service. For service composition there are a myriad of possible solutions. Figure 1 includes the Business Process Execution Language for Web Services (BPEL4WS)[ii] which has grown out of the early offerings WS Flow Language (WSFL) (IBM)[iii] and XLANG (Microsoft)[iv] (an extension of the W3C's Web Services Description Language (WSDL)).

Academic research into WSs is seeking to provide compatibility and sufficient flexibility to support the emerging commercial standards while addressing current shortcomings in the 3rd and 4th layers. Since current technology only supports syntactic and static description and composition of WS having agents automatically find, compose and execute services is not a current reality.

2.2. Semantic description of Web Services

WSDL, SOAP and UDDI are seen as steps in the right direction but ones that will fail to achieve the goals of improved automation and interoperability, because they rely on *a priori* standardisation and require humans in the loop [9]. To support automated reasoning, knowledge representations (such as markup languages) will be needed that express both data and rules for reasoning. Ontologies will be used to enable definition and comprehension of meaningful content. These are the concerns of the Semantic Web community. Additionally, agents will be needed to interpret the content and transform user requests into optimized delivered solutions.

The ability to dynamically locate and compose Web services based on their semantic description will rely on the richness of the description and the robustness of the matching techniques used. The most significant work that has been done to describe Web services has been conducted by the DAML-S coalition [16].DAML-S is built on the AI-based action metaphor where each service is either an atomic/primitive or composite/complex action. Knowledge preconditions and knowledge effects are handled via the inputs and outputs of the Web service [10]. The matching of service providers and service requesters via semantic descriptions of the services are key goals of this work. DAML-S uses the DAML+OIL specification language (which extends the weak semantics of RDF(S)) to define four upper level ontologies that can be specifically used to describe Web services. The Service ontology is essentially a means of linking the three other ontologies that contain the what (ServiceProfile), the how it works (ServiceModel) and the how to use (ServiceGrounding). Matching is typically done at the Profile level. Execution monitoring is supported via the ServiceModel, also known as the Process Model. The ServiceGrounding definition maps the DAML-S Profile and Process models to a WSDL definition of the service. To provide further compatibility with other WS standards, each DAML-S parameter may be mapped to a UDDI tModel. In section 4.2 we use the DAML-S Profile and Grounding descriptions to configure a design artifact.

2.3. Agents and Web Services

To realize the potential of agents to manage interactions with Web services a number of research efforts are under way to bring semantics to Web service descriptions that will sit at layers above what is being offered commercially. A number of approaches (e.g. [5], Racing[v]). have been offered to provide Web services with agent-like behaviour through the use of agent wrappers. [6] use wrappers so that web sources can be queried in a similar manner to databases. Alternative agent-based approaches to Web services are provided by [7] and SWORD [11] who offer

model-based approaches and deductive reasoners to derive a composition. [17] use construction scripts and composite logic to define how the services in a component can be combined, synchronised and coordinated. Typical of many approaches to composition, these approaches focus on the latter half of the system development life cycle. In [11] and [10] the goal is to determine if a set of services fulfils the specification. In all three they use a reasoner to derive a plan.

This paper seeks to fill a gap in the current work by offering an approach that is truly automatic and spans the whole system development lifecycle from requirements specification to system execution. The building blocks are Web services. The emerging DAML-S standard is used as a description language to reason about WS. The main question to be addressed in this paper is whether DAML-S descriptions of web-services offer enough structure for automated configuration by the Agent Factory.

3. Agent Factory

In this section we provide an introduction to the Generic Design Model and the concepts of a component and template.

3.1 Generic Design Model

The configuration process of a software agent in the (re-)design centre is based on the Generic Design Model (GDM) presented in [2]. In short, the assumption behind this model is that both requirements and their qualifications, and the description of an artefact evolve during a design process. E.g., in practice often not all initial requirements can be satisfied. The artefact is designed to satisfy sets of these requirements. Design choices are influenced by high-level strategies, chosen on process objectives (e.g. deadlines, resources). As shown in Figure 2 this knowledge-based model of design distinguishes reasoning about requirements and their qualifications (Requirement Qualification Set (RQS) Manipulation), reasoning about the design artefact (Design Object Description (DOD) Manipulation), and reasoning about the design process itself (Design Process Co-ordination).

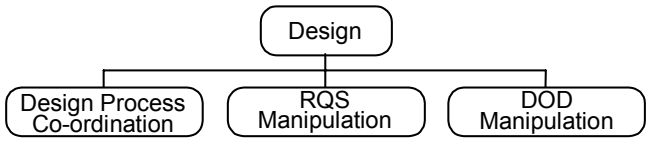

Figure 2. Main processes in GDM

The input and output of all four of these components is defined, together with the level of reasoning (meta-level) to which they pertain. Information exchange between components and potential control structures is also specified by the model as are the necessary control structures and a generic design ontology.

Design in the Agent Factory is conducted at two levels: the conceptual and implementation levels. Adaptation of an agent can involve redesign at both levels and thus requires the mapping between the two levels to be specified explicitly. The operational level includes implementation detail needed by the assembly process.

3.2 Components and Templates

A component has an interface which describes the input and output data types. Less conventionally, our components also contain slots to regulate the components configuration. Components themselves may be data types and thus may also have their own slots. Components are connected via the slots. Slots define an interface and what type of component or data type may be inserted. The use of slots provides a 'static' architecture for the agent. Templates are skeletons of components.

To support automatic agent adaptation we use two types of annotations: ontologies and co-ordination patterns. Ontologies are used to provide a shared understanding of concepts and relationships between concepts. Coordination patterns define the temporal sequence and dependencies between processes that combine to form a task. Annotations are associated with components and data types and may themselves be composed.

4. An Example

This section we provide and example of how the Agent Factory may be used for the composition of WSs in a specific domain. Section 4.1 describes the scenario. Section 4.2 follows a sample design trace of the configuration process.

4.1. The scenario

The example illustrates how the AF can be used to configure WS to create a portal containing bibliographic data.[vi].

The task of creating a portal from a given set of BibTex files is carried out by a set of web-services. First, each BibTex file is converted to RDF(S) using the BIB2RDF service then saved in a web-accessible RDF(S) repository and query engine, Sesame[vii], by the service ISESAME. The merger of all available data most often results in redundancies as different owners of the bibliographies use syntactically different resources to denote the same author. To deal with this issue the sets of redundant resources are labelled with the *sameIndividualAs* DAML tag. The task of finding and labelling redundant authors is performed by the SIA (SameIndividualAs) service. To determine the redundancies: all data is extracted from Sesame with the service ESESAME and sent to the SIA service. The results and extracted data are reinserted into Sesame using ISESAME. Finally, portal creator software creates the portals of publications by querying Sesame.

Table 1. Requirements

ID	Description
rq_i1	Create input for portal creator p_1
rq_i2	Input I_1 is generated from references in BibTex files

The initial requirement set is formulated for the design process is depicted in Table 1 Rq_i1 states that the user wants to use portal creator p_1 to create a portal on references BibTex files. This means that: the input I_1 of portal p_1 needs to be created from multiple BibTex files (rq_i2). Table 2 states the resulting additional requirements. Portal p_1 accesses the information for the portal creation from a Sesame repository (c1), which must contain references (c2), and p_1 should be able to access this information without worrying about authors being referenced differently (c3).

Table 2. Requirements of portal creator p_1

ID	Description
c1	Input I_{portal} must be in a SESAME repository
c2	Input I_{portal} contains set of references
c3	Input I_{portal} has one unique identifiers for each author

Some details that arise within the trace are in sequence:

- Sesame can handle double identifiers for the same instance if they are marked as being equal, this functional property is also stated in ISESAME.
- The input for ISESAME specified in its Profile is *data*, and *references* are a subtype of *data*. Note this relation is expressed in the ontology provided for this purpose.
- The input for ISESAME is specified in its Grounding as *rdf-stream*, which is no subtype of *data-stream.*
- A pre-condition of ISESAME is that its input needs to be tagged with sameIndividualAs before it can handle double identifiers.
- The output of is SIA specified in its Profile as *equal authors*.

4.2 An example of design

As described in Section 3.1, the Agent Factory uses the Generic Design Model as the basis for the design process. In this example reasoning about the design process (DPC), reasoning about requirements and their qualifications (RQSM), and reasoning about the design object description (DODM) are separated. Only the first part of the design trace is given. The design starts after the requirements and the constraints on portal creator p_1 have been communicated to the design process.

4.2.1. Step 1

DPC: The design process is started. The general strategy to be followed is a top-down approach: to identify a component that performs the required functionality.

RQSM: A relevant set of requirements must be compiled from the total set of requirements. The requirement rq_i1 to create input for portal p_1 is generalised to the requirement rq3. And c1, and c3 are combined to formulate requirements rq4 and rq5:

rq3 aggregate information in repository Rep_1
rq4 Rep_1 is a SESAME repository
rq5 Rep_1 identifies same instances with single identifier

This set of requirements is passed to DODM.

DODM: The first structural aspect considered is components. Functionally a web-service is sought that can store data in SESAME, and handle double identifiers for the same instance if they are marked as being equal. This functionality is covered by the web-service ISESAME. In the DAML-S profile the service category states that it stores data in a SESAME repository, and the repository can handle the DAML:sameIndividualAs-tag for identifying double instances.

4.2.2. Step 2:

DPC: Component for fulfilling requested functionality is found. Integrate this component for data-exchange.

RQSM: The relevant requirements on the data-exchange is rq_i2. This requirement is refined in rq6 and rq7.

rq6 Input are references
rq7 The input are BibTex files

The set of rq6 and rq7 are passed to DODM.

DODM: This step focuses on the structure data types. The input and output on both levels of abstraction of the component ISESAME are given in figure 3. In this figure functionality is shown with ovals for descriptions on the Profile-level, and the operational service is displayed in rectangles. On the conceptual level the data exchange poses no problems. ISESAME expects as input parameter in the DAML-S Profile *data*, which is a superclass of *references*.

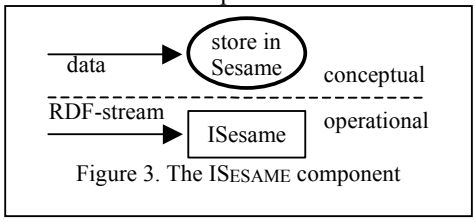

Figure 3. The ISESAME component

At the operational level there is, however a conflict. ISESAME expects an RDF-stream as input, specified in the DAML-S Grounding. However, rq7 states that the input should be BibTex files. BibTex is not of type *RDF-stream*. Therefore, to be able be used as input for ISESAME, the BibTex files should be translated into RDF. The web-

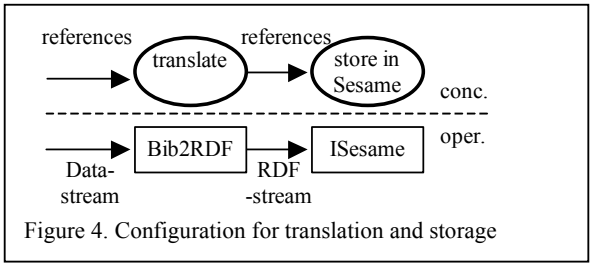

Figure 4. Configuration for translation and storage

service BIB2RDF is retrieved and included in the configuration. This web-service takes care of the translation at the operational level. In Figure 4 the result of this alteration is shown.

4.2.3. Step 3

DPC: Continue further integration of the components.

RQSM: Other requirements for checking the composition are temporal aspects. The requirement rq_i2 states that the input for the portal are gathered from multiple BibTex files. This is included in requirement rq8.

rq8 Input consists of multiple files

DODM: This step focuses on co-ordination patterns. For the creation of the portal multiple BibTex-files need to be aggregated. Therefore BIB2RDF and ISESAME need to be activated in sequence multiple times. This step results in a control construct (not depicted).

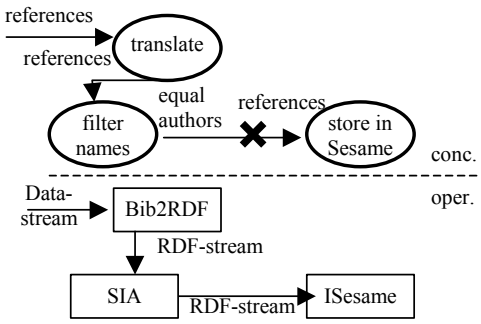

Figure 5. Configuration with error on conceptual data exchange

Further reasoning on behaviour, remaining preconditions and effects are checked for conflicts. There is one remaining conflict with respect to ISESAME, ISESAME has an additional pre-condition: to handle double instances, its input has to be tagged beforehand with the DAML:sameIndividualAs-tag. There is one web-service, which adds these tags for similar persons: SIA. This service needs to be integrated within the composition. Based on the operational in- and output, this service is activated between the BIB2RDF and ISESAME webservice.

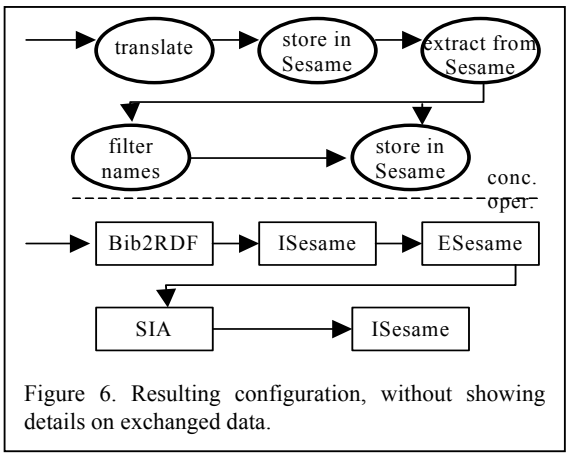

Figure 6. Resulting configuration, without showing details on exchanged data.

However, this results in a conflict on data exchange at the conceptual level. SameIndividualAs does not produce *references* as output, but *equal authors*, as shown in Figure 5. This difference does not show when only considering the XML-datatypes in the Grounding document. The solution to this problem involves multiple steps, which are not further elaborated. The resulting configuration is given, without the information flow for simplicity, in Figure 6. In this configuration, the references are translated and stored in the Sesame repository, until all files are handled, then the double author-names are filtered. The tags on equal author-names and the references are then stored together in a Sesame repository, this is the input for the portal as was requested by the user.

As shown, reasoning on function, data and behaviour is possible using DAML-S descriptions.

5. Discussion and Conclusion

Reasoning about requirements and configuring a set of components to satisfy those requirements is a novel approach to the configuration of web services. Most other work is based on workflow modeling and involves prespecification of a order and combination of WSs by a human making dynamic composition based on changing requirements impossible. Research using reusable components and patterns is not, however, unique to our work. The work by [7], which is also called the Agent Factory, is based on the notion of design patterns to assist the design of multi-agent systems. They have developed the PASSI methodology and an extended-UML CASE tool to help human designers design an agent. In the analysis/design phase, sequence diagrams are used to model protocol descriptions and class diagrams and OCL constraints are used to specify agent interactions and the knowledge agents have. The various diagrams may be compiled to generate an agent skeleton, database of patterns, reports and design documents. The Agent Factory allows the user to choose either the FIPA-OS or JADE platform. While there is much overlap at a superficial level between their work and ours, their approach aims to support developers to design agent systems while our approach is to automatically design agents. The use of the AF for Web services is a further distinguishing feature.

We note the following issues still to be resolved:

- The handling of parallel processes. We have only provided a solution which sequentially activates WSs. DAML-S does not have a means to express coordination of multiple services; DAML-S can only express control patterns within one service.
- The definition of complex services. While processes may be composed and described in the process model using DAML-S, the top level concept is a service and thus a set of services and the relationships between them cannot be

described. Szyperski [15] identifies that, today, services are almost completely self-contained, not revealing any dependencies on other services. This limits the reusability of these web-services in different contexts.

As stated in the introduction, the goal of the work reported in this paper was to review and evaluate the assumptions upon which the AF is based. Through the example, which we are currently implementing we have shown that Agent Factory can be used to automatically configure WS. We chose to test our assumptions using WS as they have many attractive features. First, they fit in the compositional view of our AF, they can easily be treated as agent components. Second, because they employ standard web protocols for interaction they are easy to integrate at the operational level. Further, the use of a semantic language for describing components at a conceptual level is promising. Our key next steps are to complete the implementation, making modifications to the process as necessary, and to further test the approach on various alternative scenarios.

Acknowledgements

The authors wish to thank the project Semantic Web at the Vrije Universiteit (SW@VU), for the usage of their developed web-services, N.J.E. Wijngaards for his work on the Agent Factory. This work is supported by the NLnet Foundation, http://www.nlnet.nl/.

References

[1] Brazier, F.M.T., Wijngaards, N.J.E. "Automated Servicing of Agents" *AISB Journal, Special Issue on Agent Technology*, 1:1 (2001) 5-20

[2] Brazier, F.M.T, Van Langen, P.H.G., Ruttkay, Zs. and Treur, J. "On formal specification of design tasks" *In Proc. of the AAAI Workshop on Artificial Intelligence and Manufacturing: State of the Art and Practice*, AAAI Press, 1994, 30-39.

[3] Brazier F.M.T., Dunin-Keplicz B.M., Jennings N.R., Treur J. "Formal specification of Multi-Agent Systems: a real-world case" In: Lesser V (ed.): *Proceedings of the First International Conference on Multi-Agent Systems, ICMAS'95*. Cambridge MA: MIT Press, 1995. p. 25-32.

[4] Brazier, F.M.T., Jonker, C.M., Treur, J. "Principles of Component-Based Design of Intelligent Agents". *Data and Knowledge Engineering* 41 (2002) 1-28.

[5] Bryson, J., Martin, D., McIlraith, S. and Stein, L.A., Agent-Based Composite Services in DAML-S: The Behavior-Oriented Design of an Intelligent Semantic Web http://www.cs.bath.ac.uk/ jjb/ftp/springer-daml.pdf.

[6] Buhler, P. A. and Vidal, J. M. "Semantic Web Services as Agent Behaviors" In B. Burg, J. Dale, T. Finin, H. Nakashima, L. Padgham, C. Sierra, and S. Willmott, eds., *Agentcities: Challenges in Open Agent Env.*, 25-31. Springer-Verlag, 2003

[7] Cossentino, M. Burrafato, P., Lombardo, S. and Sabatucci, L. 'Introducing Pattern Reuse in the Design of Multi-Agent Systems". *AITA'02 w'shop at NODe02* - 8-9 Oct. 2002 - Erfurt, Germany.

[8] Gamma, E., R. Helm, R. Johnson, J. Vlissides (1995). Design Patterns. Addison Wesley

[9] Lassila, O. "Serendipitous Interoperability", In Eero Hyvönen (ed.): *The Semantic Web Kick-Off in Finland - Vision, Technologies, Research, and Applications*, HIIT Publications 2002-001, University of Helsinki, 2002

[10] McIlraith, S. and Son, T., Adapting Golog for Composition of Semantic Web Services, Proceedings of the Eighth International Conference on Knowledge Representation and Reasoning (KR2002), Toulouse, France, April, 2002.

[11] Ponnekanti, S.H. and Fox, A. "SWORD: A Developer Toolkit for Web Service Composition" *In Proc. of 11th WWW Conf. (Web Eng. Track)*, Honolulu, Hawaii, May 7-11, 2002.

[12] Sabou, M., Richards, D. and van Splunter, S. An experience report on using DAML-S *Workshop on E-Services and the Semantic Web*, Budapest, Hungary, May 2003.

[13] Splunter, S. van, Wijngaards, N.J.E., Brazier, F.M.T., "Structuring Agents for Adaptation" In Alonso, E., Kudenko, D., Kazakov, D. (eds.) *Adaptive Agents and Multi-Agent Systems*, LNAI 2636, Springer-Verlag Berlin. 2003

[14] Staab, S., Benjamins, R., Bussler, C., Gannon, D., Sheth, A. and van der Aalst, W., Web services: Been there, Done that? IEEE Intelligent Systems, Trends & Controversies, 18(1), Jan/Feb 2003

[15] Szyperski, C. *Component Software - Beyond Object-Oriented Programming* – 2nd ed., Addison-Wesley & and ACM Press.

[16] The DAML Services Coalition "DAML-S: Web Service Description for the Semantic Web", *In Proc. of The First Int. Semantic Web Conference (ISWC)*, Sardinia (Italy), June, 2002.

[17] Yang, J. and Papazoglou, M. "Web Component: A Substrate for Web Service Reuse and Composition" *In Proc. of the 14th International Conference on Advanced Information Systems Engineering (CAiSE02)*, May, Toronto, LNCS, Vol. 2348, p21-36, Springer, 2002.

---

[i] www.stencilgroup.com/ideas_scope_200106wsdefined.html
[ii] http://xml.coverpages.org/bpel4ws.html
[iii] www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, 2001
[iv] www.gotdotnet.com/teaml/xml_wsspecs/xlang-c/default.htm
[v] http://www.zsu.zp.ua/racing/
[vi] See the SW@VU project at http://www.cs.vu.nl/~macklein/SW@VU/
[vii] http://sesame.aidministrator.nl