

Configuring Web Services, using Structuring and Techniques from Agent Configuration

S. van Splunter¹, M. Sabou¹, F.M.T. Brazier¹, D. Richards²

¹ *Department of Computer Science
Vrije Universiteit Amsterdam
Amsterdam, The Netherlands*

<sander, marta, frances>@cs.vu.nl

² *Department of Computing
Macquarie University
Sydney, Australia*

richards@ics.mq.edu.au

Abstract

This paper explores the use of an Agent Factory for the composition of web services. Previous work proposed a structuring approach for automated reconfiguration of agents by an Agent Factory. The question is whether the same approach can be applied to web service composition, i.e. whether DAML-S descriptions of web services offer enough structure for automated configuration by the Agent Factory. An example trace of the Agent Factory for configuration of DAML-S web services illustrates this approach.

1. Introduction

The increasing proliferation of web services offers a means of addressing the ever-increasing demand for applications. The real value of web services is in their composition. Web service composition is not just an alternative to application development, but a means of reducing the application backlog problem providing new and value-added functionality. It is encouraged by three facts: many services are moving online; WS conform to the HTTP protocol which makes integration easier, and many independent providers have related services that need to be combined to satisfy user requirements [1].

A myriad of products and solutions support composition of web services, many of them coming from earlier workflow and business process engineering approaches (e.g. [2]). However, workflow approaches do not handle dynamic and distributed composition of web services: workflow management systems (WFMS) do not all share the same workflow syntax and semantics, and do not support changes to workflow definitions [3]. Commercial solutions tend to be tools with a supporting methodology to capture the process flows from a human designer. Some techniques (e.g. [4, 5]) that claim dynamic composition, rely on the requestor and provider having already been matched. Composition involves determining which implementation of a service is most appropriate based on the constraints specified by the user. These approaches can be classified to be, at most, semi-automatic.

A number of approaches (e.g. [6], Racing¹) provide web services with agent-like behaviour through the use of agent wrappers. [7] 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 [8] and SWORD [1] who offer model-based approaches and deductive reasoners to derive a composition. [3] use construction scripts and composite logic to define how the services in a component can be combined, synchronised and co-ordinated. Typical of many approaches to composition, these approaches focus on the latter half of the system development life cycle. In [1] and [8] 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 web services. The main question addressed in this paper is whether DAML-S descriptions of web services offer enough structure for automated configuration by the Agent Factory.

Section 2 considers the two main technologies involved: web services and the Agent Factory. Section 3 presents our ideas about how to combine the two technologies. An example of the use of the Agent Factory for web service composition is given in section 4. Discussion and future work are given in the final two sections.

2. The two technologies

This section introduces the two technologies combined in our work: web services and the Agent Factory.

2.1. Web services

A web service (WS) is a (self-contained) software

¹ <http://www.zsu.zp.ua/racing/>

component that allows access to its functionality via a web interface. WSs communicate by employing established protocols for message transport and encoding. Industry efforts have identified major issues related to WSs and have developed a set of proposals for each. Figure 1 (adapted from [9]) depicts the WS architecture: each layer corresponds to the main areas within the WSs field and includes the effort/s that appear to have the most support as a standard in that particular area. The academic work reported in this paper is positioned in *italics>. At the transport level, WSs rely on traditional web protocols. The Simple Object Access Protocol (SOAP) is an XML-based communication protocol that allows exchange of data via typed messages 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) standard service repository. For specifying service composition there are many possible languages, including the Business Process Execution Language for Web Services (BPEL4WS) [10] depicted in Figure 1. BPEL4WS has grown out of two earlier languages: Web Services Flow Language (WSFL) (IBM) [11] and XLANG (Microsoft) [12]. The influence of its developers makes its acceptance as a standard likely.*

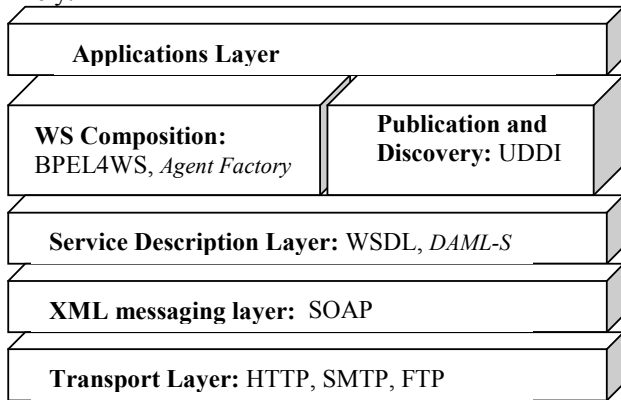


Figure 1: Overview of WS technology

SOAP, WSDL, UDDI, and BPEL4WS are the standard combination of technology to build a WS application. However they fail to achieve the goals of automation and interoperability because they rely on *a priori* standardisation and require humans in the loop [13]. To support reliable, large-scale interconnectivity of web services by software, computer-processable semantics, which include the properties, capabilities, interfaces, and effects of the service [8] are needed.

The Semantic Web community proposes the use of semantics for WSs for this purpose. There are a number

of efforts in this area, but the work gaining the most attention is an approach developed by a large coalition of researchers, known as DAML-S[14].

DAML-S facilitates automatic discovery, invocation, composition, interoperation and monitoring of WSs through their semantic description. It is a DAML+OIL ontology conceptually divided into three sub-ontologies for specifying *what a service does, how the service works* and *how the service is implemented*. Accordingly, each DAML-S description has three major parts: the Profile, Process and Grounding.

2.2. Agent Factory

An Agent Factory (AF) [15] is a service for automated (re-)design of software agents. An Agent Factory service distinguishes three main sub-processes: 1) (Re-)design; 2) Building block retrieval; and 3) Assembly. The *(Re-) design process* produces a specification of an agents configuration, given a set of qualified requirements. *Building block retrieval* is based on queries with functionality, behaviour and state. In *Assembly*, operational code is assembled on the basis of an operational configuration specification. This paper focuses only on the (Re-)design process, also referred to as the Design process.

This Design process is one of configuration based on the Generic Design Model (GDM) as presented in [16]. 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.

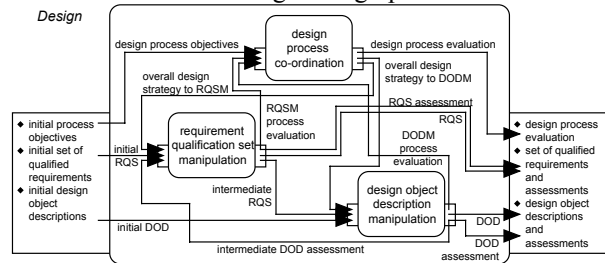


Figure 2. Composition of processes of the design process in the generic model of design.

Figure 2 shows one level of composition of the processes distinguished within the model, and the types of input and output involved. The refinements of these three processes are not further depicted (see [16] for a description of a formal specification including details on additional process composition, information flow, control flow, generic information types and generic knowledge bases). The left hand side describes the input information to the design process; the right hand side describes the output information. The design process is shown to be composed of three sub-processes: design process co-ordination, requirement qualification set manipulation,

and design object description manipulation. The process Design Process Co-ordination (DPC) co-ordinates the design process itself by issuing information related to overall design strategies on the basis of progress reports of the manipulation components and given design process objectives. An example of a design process objective that needs to be guarded by DPC is, e.g., available time or budget. The process Requirement Qualification Set Manipulation (RQSM) manipulates sets of requirements (RQS), on the basis of an overall design strategy, information from Design Object Description Manipulation (DODM), and given sets of qualified requirements. Requirements can be, e.g., refined, added, or receive another qualification to focus DODM. The process DODM manipulates descriptions of design objects (DOD), on the basis of an overall design strategy, information from RQSM, and given design object descriptions. Manipulations of a DOD can be additions, modifications, or deletions of domain object information.

One important assumption on which the design model and thus the Agent Factory is based, is that the artefact, i.e. the agent, has been designed to be re-designed. This implies three things [17]. First, agents have a compositional structure with reusable parts, building blocks. Second, at least two levels of description of agent configurations are defined: conceptual and operational. In the Re-design process of the Agent Factory operational building blocks include implementation details needed by the Assembly process to create realisations of conceptual building blocks. Third, there are ontologies to describe the functionality, behaviour, and state of agents and their components. The assumption that an agent must have a compositional structure, relies on the availability of compositional models of agents, e.g., ZEUS [18], the GENERIC AGENT MODEL [19], and the JADE AGENT MODEL [20].

Descriptions of an artefact in which function, state, and behaviour are specified translate directly to the *components*, *data types*, and *co-ordination patterns*. These structures are essential for the Agent Factory [17]. *Components* refer to the (active) processes distinguished within an agent (which may in turn be composed). They are modelled as building blocks, which, less conventionally, can also contain open slots. Slots for components define constraints for the functionality of the component that may be inserted and an interface. Components specified with building blocks can therefore be partial specifications of a process. Examples of such partial specifications are agent models.

Data types refer to the information exchanged and manipulated by components. Each data type represents a specific piece of information. Data types can be either primitive or composed. In the Agent Factory data types are also modelled as building block possibly with open

slots. Slots for data types define an interface and constraints with respect to the required semantics.

Co-ordination patterns are used to define the temporal sequence and dependencies between tasks. Co-ordination patterns specify the activation of tasks, and information flow between tasks. The tasks specified in a co-ordination pattern may be directly mapped onto components, but this is not necessarily the case. A simple example of a co-ordination pattern is one that specifies the behaviour of a composed component: all of the tasks in a co-ordination pattern are directly mapped to sub-components. A task may, however, involve a number of components that are not part of the same composed components. This requires the specification of a more intricate mapping. Co-ordination patterns can be specified for both conceptual and operational building blocks. For further motivations and background to this structuring approach of the artefact, see [17].

3. Composing WS with the Agent Factory

The previous section distinguishes three processes within the AF. This section focuses on the design process: the configuration of web services, given the relevant building blocks. Building block retrieval and assembly are not addressed.

The AF operates on building blocks (concerning components, data types and co-ordination patterns). Web services have a very similar metaphor: they are components that operate on data types and can be composed using composition patterns. However, because WSs are self-contained, there is a much stronger link between its elements than in the case of the AF. Each web service is in fact a component, and its inputs and outputs are data types. The internal working of the web service or a specification of combination of multiple web services is a co-ordination pattern. This implies that a web service is fully described by the combination of the three available types of building blocks. Because WSs do not contain open slots, in our focus, they are therefore simple building blocks. The AF and WSs employ both a conceptual and an operational description. DAML-S is used to specify conceptual building blocks and WSDL to express operational level details. DAML-S contains a Profile, a Process and a Grounding.

The *Profile* describes what the service does. The primitives for the functional description of the service consisting of Inputs, Outputs, Preconditions and Effects (IOPE's) are most relevant in the context of this paper.

The *Process* presents the internal working of the service in terms of the internal processes, their process model and the internal data-flow. It is envisioned that this information is useful for monitoring the execution of the service. The Process describes the IOPE's of the service from a different perspective, but naturally links between the Profile and the Process of the model exist. The Profile

IOPE's refer to the corresponding IOPE's in the Process. Note that both parts of the description augment the described elements with domain level concepts pre-defined in an external ontology. Taken together they represent the conceptual level description of the service.

The *Grounding* specifies the operational level details of the service by linking the conceptual level descriptions to the WSDL description of the service. Finally, the WSDL file contains the implementation details of the service such as message formats and access protocols.

Table 1 summarises the DAML-S primitives used to describe AF components, data types and co-ordination patterns.

Table 1. The relation between AF artefact structures and DAML-S concepts.

AF artefact Structure	Related DAML-S concepts
Components	- Service (with Process, Profile and Grounding)
Data types	- IO's (in Profile, Process, and Grounding) - External ontologies
Co-ordination patterns	- ControlConstructs for CompositeProcess - Pre-conditions and Effects

Components: A DAML-S Service is a component. Its Profile describes the functionality of the service and its Process its behaviour. Its Grounding builds a bridge from conceptual to operational level. Theoretically, DAML-S allows grounding a single conceptual description to multiple operational descriptions. However, to avoid confusion, we opted for the simplified version of a one-to-one correspondence between conceptual and operational descriptions. This is a simplification also for the AF, which allows conceptual models to have more than one implementation model, and vice versa.

Data types: At a conceptual level, data types are described by the IO's in the Profile and Process model. As a DAML+OIL-based ontology, DAML-S offers "rich typing", which enables the expression of data types and the relationships between data types. At the operational level, data types are described in WSDL. Note, however, that within DAML-S the mapping from conceptual to operational data types is restricted to a one-to-one mapping.

Co-ordination patterns: DAML-S supports the description of co-ordination patterns via the Process Model, which uses a set of ControlConstructs (e.g. Sequence, Split, Choice) to define the internal control of the internal processes. Pre-conditions and Effects (PE's) can be used to express dependencies between web services. Within the Profile description, only PE's concerning the usage of the whole service are specified. The Process description can describe specific PE's that occur during the use of the web-service. Thus the Process

description of a composite process using ControlConstructs and PE's, is a single co-ordination pattern that has been instantiated and is only useful possibly for matching or monitoring purposes but not for dynamic composition.

In summary the DAML-S WS description language is sufficiently expressive for specifying conceptual and operational building blocks.

4. An example

This section illustrates the use of the Agent Factory for the composition of WS, for a specific domain. Section 4.1 describes the scenario. Section 4.2 elaborates on the example design trace of a configuration process.

4.1. The scenario

The example chosen to illustrate how the AF can be used to configure WS is that of the creation of a browseable portal for bibliographic data². The bibliographic data is originally expressed in a number of BibTeX files.

The portal makes use of a set of web services (denoted in small capitals): BIB2RDF, ISESAME, SIA, and ESESAME. These WSs can be used either individually or in composition, depending on, e.g., the format of the files, or availability of the information in a Sesame repository. In the scenario presented in this section all of these WS are needed.

First, each BibTeX file is converted to RDF(S) using the BIB2RDF service, then saved in Sesame³, a web-accessible RDF(S) repository and query engine, by the service ISESAME. The merger of multiple BibTeX files most often results in implicit redundancies as different owners of these bibliographies use syntactically different resources to denote the same author. To make this information explicit sets of redundant resources on authors are identified and labelled with the *sameIndividualAs* DAML tag 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 the repository of Sesame using ISESAME. Finally, portal creator software, not represented as a separate WS, creates the portals of publications by querying Sesame.

² The services used in this example have been developed within the context of the SW@VU project, see <http://www.cs.vu.nl/~mcklein/SW@VU/>

³ <http://sesame.aidministrator.nl>

Table 2. Initial Requirements

ID	Description
rq _i 1	Create input for portal creator p ₁
rq _i 2	Input I ₁ is generated from references in BibTeX files

The initial requirement set formulated for the design process is depicted in Table 2. Rq_i1 states that the user wants to use portal creator p₁, the portal from the SW@VU-project. This means that: the input I₁ of portal p₁ needs to be created from multiple BibTeX files (rq_i2).

Table 3 states the additional requirements, resulting from the usage of portal p₁. Portal p₁ accesses the information for the portal creation from a Sesame repository (rq_{p1}3), which must contain references (rq_{p1}4), and p₁ should be able to access this information without worrying about authors being referenced differently (rq_{p1}5).

Table 3. Requirements of portal creator p₁

ID	Description
rq _{p1} 3	Input I _{portal} must be in a Sesame repository
rq _{p1} 4	Input I _{portal} contains set of references
rq _{p1} 5	Input I _{portal} has one unique identifiers for each author

Within the trace in section 4.2 decisions in the design are made based on observations of conflicts, or of details within WS descriptions. As a reference point these observations are given beforehand:

- 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*. Conceptually BibTeX files can be used as input for ISESAME.
- The input for ISESAME is specified in its Grounding as *RDF-stream*, which is no subtype of *data-stream*. Operationally BibTeX files can not be used as input for ISESAME.
- A pre-condition of ISESAME is that its input needs to be tagged with DAML:sameIndividualAs-tag before it can handle double identifiers.
- The output of is SIA specified in its Profile as *equal authors*. This implies that not whole references are returned as output.

4.2 An example of design

As described in Section 2.2, the Agent Factory uses the Generic Design Model as 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 initial requirements and the requirements of portal creator p₁ have been communicated to the design process.

4.2.1. Step 1

DPC: The design process is started. The general strategy 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 p₁ is generalised to the requirement rq6. And rq_{p1}3, and rq_{p1}5 are combined to formulate requirements rq7 and rq8:

- rq6 aggregate information in repository Rep₁
- rq7 Rep₁ is a Sesame repository
- rq8 Rep₁ 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, which can handle the DAML:sameIndividualAs-tag for identifying double instances.

4.2.2. Step 2:

DPC: Now the component for fulfilling requested functionality has been found. This component needs to be integrated for data-exchange.

RQSM: The relevant requirement on the data-exchange is rq_i2. This requirement is refined to rq9 and rq10.

- rq9 Input are references
- rq10 The input are BibTeX files

The set of rq9 and rq10 are passed to DODM.

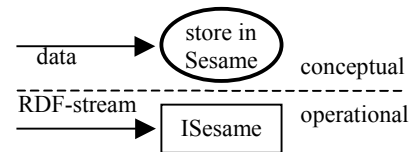


Figure 3. The ISESAME component

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*.

On the operational level there is, however a conflict. ISESAME expects an RDF-stream as input, specified in the DAML-S Grounding. However, rq10 states that the input should be BibTeX files. BibTeX is not of type *RDF-stream*. To be able to be used as input for ISESAME, the BibTeX files should therefore be translated into RDF. The web service BIB2RDF is retrieved and included in the configuration. This web service performs the translation at the operational level. In Figure 4 the result of this alteration is shown.

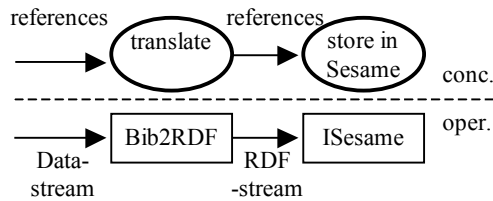


Figure 4. Configuration for translation and storage

4.2.3. Step 3

DPC: Continue further integration of the components.

RQSM: The requirement rq2 states that the input for the portal is gathered from multiple BibTeX files. This is included in requirement rq11.

rq11 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).

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 web services.

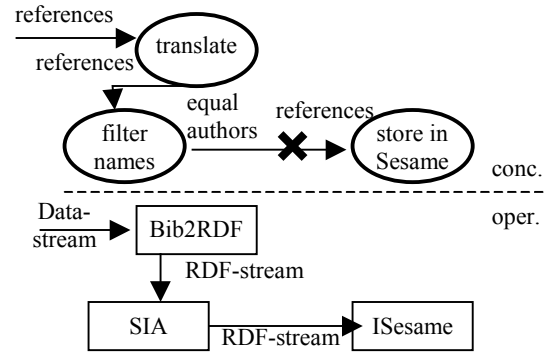


Figure 5. Configuration with error on conceptual data exchange

However, this results in a data exchange conflict 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-data types 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, after which 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 in this trace, reasoning on function, data and behaviour is possible using DAML-S descriptions.

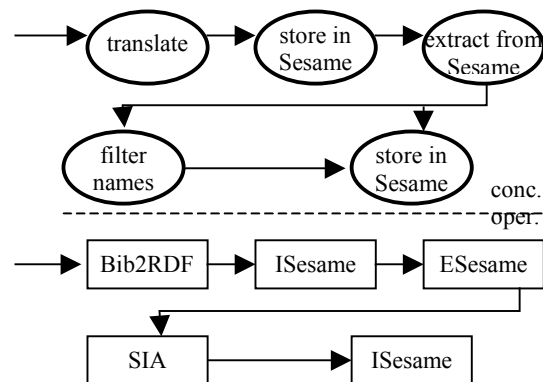


Figure 6. Resulting configuration, without showing details on exchanged data.

5. Discussion

The use of components and reusable patterns is a recurring theme in a number of research efforts. The work by [21], 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.

This paper has shown that the concept of an Agent Factory such as the one described in [15] can be used to automatically configure WS. The design model on which the Agent Factory is based, is one of configuration, in which reasons about requirements is an explicit configuration, in which reasoning about requirements is an explicit part of the design process.

WSs 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. Despite these positive conclusions there are a set of open issues to be considered.

(1) Control. In this paper, only sequential activation of web services has been considered, and not parallel activation. This is an issue not only for configurations of web services distributed over multiple servers, but also for services on the same server. DAML-S does not have a means to express co-ordination of multiple services; DAML-S can only express control patterns within one service.

(2) Complex services. Complex services, composed of atomic services, cannot be described with DAML-S [22]. (Note that processes, however, may be composed).

(3) Extensibility. Szyperski [23] 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. Further, Szyperski states that, for reuse, the context dependencies have to be made explicit. Open slots as implemented in the Agent Factory are a way to define "dependencies" and to specify interfaces. A simple example in which an implementation of a web service operationally forces an open slot is a web service that has as its input parameters the URL of the web service that it has to use. Our concern is that simply using IOPE's for specifying dependencies between web services will not

suffice to support more complex configuration tasks. As the open slot concept is new to WSs, it is not directly supported by DAML-S either. Extending DAML-S to include a property `isOpenSlot` having as domain a Profile instance and as range a Boolean value, is a possible solution. A "True" value means that the specific Profile instance must be considered as an open slot. For a component that exposes an open slot, the corresponding service will have a Profile instances marked as being an open slot (i.e. the value of `isOpenSlot` is true).

(4) Limitations of DAML-S. The use of DAML-S is not always straightforward [22]. The conceptual model underlying DAML-S is imprecise. Different parts of the language build on different metaphors (action/ function). The links between these conceptual models are poorly specified and often inconsistent. Within DAML-S there is also little reference to standard Software Engineering terminology: while basic concepts are employed (such as "function/ method"), there are no directions given about how to model more complex situations (such as parametric polymorphism). This imprecise conceptual model provided flexibility in modelling, but more often it led to confusion.

6. Future work

Our current and future work focuses, and will focus on three areas. First, to extend our experiments to more complex services and composition scenarios that require the full functionality of the Agent Factory. Second, to find a way to express complex services and composition of multiple services, elements that are necessary for exploring the full potential of our design methodology. Finally, to explore the use of composition languages (BPEL4WS, WSFL, XLANG, WSCI, BPML) within the Agent Factory. Our initial concern is that, for automated knowledge and (advanced) process composition, semantic information will be needed and that it cannot (yet) be expressed by these industry standards, however newer standards of, e.g., UDDI are moving to further integration of meta-data enabling automated web service configuration.

Acknowledgements

The authors wish to thank the project Semantic Web at the Vrije Universiteit (SW@VU), for their web services, N.J.E. Wijngaards for his work on the Agent Factory, and Stichting NLnet for their support (<http://www.nlnet.nl/>).

References

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

- [2] Narendra, N. C. "AdaptAgent: Integrating Adaptive Workflows and Multi-Agent Conversations for B2B E-Commerce" In *Proc. of International Conference on Artificial Intelligence*, Special Session on Agent-Oriented Workflow Architecture for B2B, 2001.
- [3] 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, Lecture Notes in Computer Science, Vol. 2348, p21-36, Springer, 2002.
- [4] Casati, F., Ilnicki, S. and Jin, L. "Adaptive and Dynamic Service Composition in eFlow" *HP Technical Report, HPL-2000-39*, March, 2000, <http://www.hpl.hp.com/techreports/2000/HPL-2000-39.pdf>
- [5] Tomic, V., Pagurek, B., Esfandiari, B. and Patel, K. "On the Management of Composition of Web Services" *Workshop on Object-Oriented Web Services - OOWS (at OOPSLA 2001)*, Tampa, USA, October 15, 2001.
- [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, editors, *Agenticities: Challenges in Open Agent Environments*, pages 25-31. Springer-Verlag, 2003
- [7] Knoblock, C.A., Minton, S., Ambite, J.L., Muslea, M., Oh, J. and Frank, M. "Mixed-initiative, multi-source information assistants" In *Proc. of the WWW Conference*, pages 697--707, ACM Press, New York, NY, May 2001.
- [8] McIlraith, S., Son, T.C. and Zeng, H., "Mobilizing the Semantic Web with DAML-Enabled Web Services", In *Proc. of the Second International Semantic Web Workshop (SemWeb'2001)*, Hongkong, China, May, 2001.
- [9] Van de Aalst, W. "Don't Go with the Flow: Web Services Composition Standards Exposed" *IEEE Intelligent Systems*, 18:1, 2003, 72-76
- [10] XML Cover Pages. Business Process Execution Language for Web Services (PBEL4WS). <http://xml.coverpages.org/bpel4ws.html>.
- [11] Leyman, F. "Web Service Flow Language (WSFL) 1.0", IBM, Armonk, NY, www-4.ibm.com/software/solutions/webservices/pdf/WSFL.pdf, 2001.
- [12] Thatte, S. "XLANG: Web Services for Business Process Design", www.gotdotnet.com/team/xml_wsspecs/xlang-c/default.htm, 2001.
- [13] 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
- [14] The DAML Services Coalition "DAML-S: Web Service Description for the Semantic Web", In *Proc. of The First International Semantic Web Conference (ISWC)*, Sardinia (Italy), June, 2002.
- [15] Brazier, F.M.T., Wijngaards, N.J.E. "Automated Servicing of Agents" *AISB Journal, Special Issue on Agent Technology*, 1:1 (2001) 5-20
- [16] Brazier, F.M.T, Van Langen, P.H.G., Ruttkay, Zs. and Treur, J. "On formal specification of design tasks" In *Proc. of the AAI Workshop on Artificial Intelligence and Manufacturing: State of the Art and Practice*, AAAI Press, 1994, 30-39.
- [17] 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*, Lecture Notes in Artificial Intelligence (LNAI) 2636, Springer-Verlag Berlin. 2003.
- [18] Nwana, H.S., Ndumu, D.T., Lee, L.C. "ZEUS: An Advanced Tool-Kit for Engineering Distributed Multi-Agent Systems". *Applied AI* 13:1/2, 1998, 129-185.
- [19] 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.
- [20] Fabio Bellifemine, Agostino Poggi, Giovanni Rimassa: Developing multi-agent systems with a FIPA-compliant agent framework. *Software - Practice and Experience* 31(2): 103-128, 2001.
- [21] Cossentino, M. Burrafato, P., Lombardo, S. and Sabatucci, L. "Introducing Pattern Reuse in the Design of Multi-Agent Systems". *AITA'02 workshop at NODE02 - 8-9 October 2002 - Erfurt, Germany*.
- [22] Sabou, M., Richards, D. and Splunter, S. van, "An experience report on using DAML-S", *Workshop on E-Services and the Semantic Web*, Budapest, Hungary, May 2003.
- [23] Szyperski, C. *Component Software - Beyond Object-Oriented Programming - Second Edition*, Addison-Wesley and ACM Press, ISBN 0-201-74572-0, 2002.