Automated Web Service Composition Methods and Tradeoffs

Markus Stumptner
University of South Australia,
Advanced Computing Research Centre,
5095 Mawson Lakes (Adelaide) SA, Australia,
email: mst@cs.unisa.edu.au

Abstract: This paper examines the use of constraint-based configuration for the composition of Web Services. Web Services are widely assumed to represent the basis for the next generation of flexible distributed applications in B2B E-commerce, and the composition of complex applications from individuals services has attracted much attention. We show how this composition problem can be addressed at increasing levels of semantic content embedded in the description of services, moving from purely manual composition to describing service matching as a configuration problem that can be solved using constraint-based methods. We examine the restrictions imposed by Semantic Web ontology languages and providing a succinct and high level mechanism for imposing the different boundary conditions resulting from the multilayered application environment. We give an example showing the configuration process for a simple example problem, discuss the ramifications of the full composition problem, and describe the resulting system architecture.

1 Introduction

Web Services [ACKM04] have grown to be widely perceived as the architecture that will fuel the next generation of flexible distributed applications in B2B E-commerce and other fields, with individual Web Services being composed into more complex applications. In this view of the distributed computing world, a Web Service is basically an individual component that communicates with its environment through a set of parameterized, messages typically using a web-related protocol and XML-based encodings for the messages. Web Services have promise to adress multiple different but related concerns at the same time:

- encapsulating business logic in clearly defined components
- flexibility and standardization through HTTP-based communication
- homogeneous interface can hide heterogeneous local information systems
- distributed execution of multiple web services (from potentially different provides) within the scope of the same application
- Dynamic selection of desirable candidates for cooperation from a set of offered candidates (*service matching*)

Different languages for describing Web services have been developed over the past few years; at the moment the most support candidate is WSDL (Web Service Description Language) [CCMW01]. WSDL describes a service in terms of multiple *ports*, each of which defines sets of ingoing and outgoing messages that can be used to communicate with the service through that port. A message is an XML structure that contains the parameters (called *parts*) necessary for execution of the service, or passes back the result. WSDL defines four operation modes, *notifications, one-way, solicit-respons*, and *request-response*, each of which corresponds to a particular combination of output and input messages (e.g., *solicit-response* generates an output message and receives a return message). In addition, a WSDL description gives a *binding* for a service, specifying the actual communication protocol (e.g., SOAP, HTTP, or MIME) and its settings.

The service matching counterpart to WSDL's service descriptions is UDDI (Universal Description, Discovery and Integration) [ea02b]. UDDI describes businesses in terms of physical attributes (name, address, and lists of services), plus extended attributes called *TModels* that describe services by reference to standard taxonomies such as NAICS (North American Industry Classification System). However, UDDI has been recognized to be severely limited in practice due to the fact that it solely supports keyword searches (in effect, string matching) on its attributes and does not permit any description of actual service semantics. As a result it is either bypassed or extended by all the suggested approaches below.

This is because, as is clear from the list of expectations after the first paragraph, in particular the fifth property, that individual Web services by themselves are of limited applicability. They are either only considered a building block for more complex applications that can be composed through the (possibly repeated) execution of multiple different Web services, or at the very least independent components available for distributed execution that therefore require a separate process for their identification and use. In particular the last of the five properties listed above lies at the heart of the web service composition problem. A given set of web services can be arranged in different ways, or different web services could theoretically be arranged to solve the same problem depending on particular conditions for a given application. It is at this level that WSDL descriptions are considered insufficiently powerful to capture the information required for the arranging process, and extensive research on other methods has been conducted.

Methods that have been studied so far to escape the WSDL/UDDI description bottleneck range from largely static specification methods developed on the basis of classical workflow modeling [YP02] over dedicated modeling languages [FLP$^+$03], ontology languages [MBE03], and dedicated Web service description languages [The01, NM02] to agent-based approaches [vSSBR03]. Our concern lies mostly in examining the nature of the composition problem and identifying to what degree configuration methods can be put to use.

In the next section we will briefly discuss the basic infrastructure for Web service composition, and then examine thes different approaches and their shortcomings. In Section 3.4 we show how service matching generation can be described as a configuration problem and solved using constraint-based methods, circumventing the restrictions imposed by Semantic Web ontology languages and providing a succinct and high level mechanism for imposing the different boundary conditions resulting from the multilayered application environment. We give an example showing the configuration process for a simple example problem, discuss extensions including full service composition in Section 3.6 and finally discuss architectural issues in Section 3.5.1.

2 Infrastructure for Web Service Composition

Conceptually, two subtasks of the WS composition task can be distinguished, the actual logical arrangement and connection of the different services (also known as *choreography* or *orchestration* [KSR04]), and the selection of the particular services that are suitable to participate in the application in situations where multiple services are being offered (also known as *matching*). Generally, choreography solutions assume the existence of a human developer who uses some high level (often graphical) language that explicitly specifies the operations called at a particular point in the sequence. We use *composition* here to refer to approaches where a significant part of this process has been automated.

For the purpose of this paper we start by examining a simplified version of the WS combination model proposed by [YP02]. We refer to the parameters of the output messages of a service s as $o(s)$ and to the parameters of the input messages as $i(s)$.

Definition 2.1 *Given a set of services* $S = s_1, \ldots, s_n$, *with* $O = \bigcup_{o(s)}$, $I = \bigcup_{i(s)}$ *a* composite service *is a tuple* $< S, M >$, *and the partial function* $M : O \mapsto I$ *is the* *parameter mapping.*

The definition of [YP02] also permits specifying a *mode* for composition that indicates whether alternate services that fit the description of S are executed in parallel or sequentially (in case one fails). For space reasons we do not address this point or specify at this time *how* particular output values are mapped to input values in other messages, merely that they are. The mapping is partial because individual parameters may have default values or go unused, and it could involve more complex transformations (thus incorporating the message synthesis and message decomposition of [YP02]).

2.1 Web Service Composition Languages

In the last years, a number of new conceptual languages for describing application behaviours have been developed, mostly with a view towards modeling business processes and web services. So far no clear standard language has emerged, although BPEL4WS seems to have the strongest industry base.

UML EDOC : The UML Profile for Enterprise Distributed Object Computing (EDOC) was released in 2002 as a MOF (UML Meta Object Facility) compliant framework by the OMG. It has an expressive graphical notation but the semantics for its lifecycles are limited and ambiguous.

XPDL: The Workflow Management Coalition (WfMC) has developed an XML-based Process Definition Language (XPDL) for supporting the interchange of process descriptions between different workflow systems. It is rich in modelling concepts but there is no graphical representation.

Web service languages: Most of the released web flow languages are based on XML and are built on the top of WSDL. Remember that WSDL describes only static interfaces. The Business Process Execution Language for Web Services (BPEL4WS) combines the concepts of Microsoft's XLANG and IBM's WSFL. It allows the description of life cycles and provides a message correlation model.

ebXML: ebXML offers a standard for describing business processes and for choreography of business transactions in business collaborations. It focuses on the exchange of business messages, collaboration agreements and collaboration profiles. ebXML uses UML for business process specification, but limits itself to class diagrams, which are not intended for the direct creation of ebXML business process specifications [Tea01].

Common to all these languages is that they assume a fixed assignment of the individual web services to the cooperation described, and that the matching task has already been performed (more likely manually than automatically). Formal criteria for correct matchings under these conditions have been described in [YP02], but are restricted by the limited information available. A service S consists of a set of unspecified contents C, activites A, and properties P.

Definition 2.2 *Activity a of S is* compatible *with a' of S' based on equivalence of their respective pre- and post-conditions and the assumption that the input and output sets of a are (respectively) a subset of those of a' (co-variance, or observation consistency [SS02]), and S is compatible with S' if any operation of S is compatible with some of S', and the content of S is a subset of that of S'. S* conforms *to S' if their contents overlap and there is one pair of operations $a \in A$, $a' \in A'$, so that the output of a is compatible with the input of a'.*

Thus, conformance is a necessary condition for a correct outcome but not a sufficient one. We continue with an examination of the actual approaches that have been proposed to address the issue.

3 Matching and composing web services

3.1 Automatic Matching Support

In what is called an *ontological workflow approach*, the POESIA system [FLP$^+$03] describes Web services in terms of "ontological coverage" and relates different services through a separate "process framework" (which describes dependencies and therefore a partial time ordering) and an aggregation hierarchy. Both the correctness of the aggregation hierarchy and the appropriateness of the process framework (e.g. deadlock freeness) can be automatically checked based on the data dependencies existing bewtween differen services. Ontological coverage is used to restrict choice of services to preserve consistency between the different services' data needs.

The SWORD system [PF02] is a rule-based expert system for web service conposition that uses a special-purpose planning approach to determine whether a particular service can be composed from subsidiary ones. Lower level services are chained together by testing whether the preceding service's postcondition implies the following service's precondition.

3.2 Ontology-based matching

In [MBE03], Medjahed et al. propose a composition method based on identifying a fixed set of compatibility criteria. Web services follow the standard WSDL model, but no less than 17 extra descriptors are used in the model.

- For parameters, this extended modeling framework records, where applicable, the measurement units associated with parameter values, and also specifies a *business role* for each parameter.
  Each operation has an associated *purpose, category*, and *quality*. Each of these three again consists of three parameters.
- Purpose and Category are described by "function" value taken from some taxonomy (e.g., RosettaNet or cXML) for Purpose and a "domain" for category, plus a list of synonyms, and possible textual qualifiers ("specialization").
- Quality is described by *fees* (charges for executing the service), *security* (secure transmission yes/no), and *privacy* (a list of those parameters that are not shown to external entitites). Effectively all these attributes are represented as strings, but given their limited choice from a specific taxonomy, can also be thought of as basically domain-specific scalar values.

Finally, each Web service has a set of potential bindings. Based on these definitions, Web services are determined as being composable in a number of steps that check whether are composable in terms of

- mode, i.e., whether their message types are complementary (e.g., one is of type `request-response`, and the other of type `solicit-response`),
- the parameters of corresponding messages (with one being possibly a subtype of the other)
- purpose and category (with their respective Domains the same or synonyms, and one's specialization a subset of the other)
- finally, quality. This is checked individually for each property, e.g.,the offered fees must be at least as large as the requested ones.

In addition the information provided describes whether any of the web services is supposed to call operations of another one, in terms of a directed graph called the *stored composition template*. The actual composition is required to represent a subgraph of the stored template. A special purpose language and matching algorithm operate on this representation. The algorithm incorporates in hardcoded fashion the relevation comparison operators for the different descriptors, i.e., *purpose_compatible, category_compatible, quality_composable, and message_composable*. In case particular criteria are not fully satisfied, they are weighted to the degree of fulfillment.

Overall it can be said that while this approach attempts to be comprehensive in terms of the attributes and powerful in terms of the algorithm, its flexibility is limited. The ontology referred to in the method's name is *one fixed* ontology that all services have to satisfy; there can be no extensions.

3.3 OWL-S

OWL-S (formerly DAML-S) [ea02a] is an ontology (or a set of three related ontologies) for what its authors call the "semantic description" of Web services. The description is divided into *Service Models*, which describe the way a Web service operates, *Service Profiles*, which describe the advertised interface of a service (considered synonymous with "what the service does"), and the *Grounding*, which maps the service to a specific WSDL description (commonly referred to as "how to use the service", although it appears that to use the service one needs to be aware of the other two models as well).

A Service Model has associated Processes that are atomic, composite, or "simple" (a category that basically serves the information hiding of internal process details – a simple process can map to an atomic or composite one). Composite processes can be constructed from simple or atomic ones using various operators (sequence, concurrent, while, etc.). Atomic processes correspond to WSDL operations, and their inputs and outputs depend on the type of WSDL operation represented. In addition to inputs and outputs, services are described by their preconditions and effects, and the complete semantic description of a service given by OWL is referred to as its IOPE. A significant number of proposals for service matching in the OWL-S context have been published in the last years.

Paolucci [PKPS02] provides a list of requirements for a matching engine:

1. support "flexible semantic matching" between advertisements and requests "on the basis of the ontologies available"
2. minimize false positives and false negatives
3. the requesting service should have some control over the amount of matching flexibility
4. the matching engine should encourage advertiser and requesters to be honest at the cost of either not being matched or being matched inappropriately
5. the matching process should be efficient

Of these, criterium 1 basically states matching should be as good as possible with the information available, and 4 merely means that matching needs to be correct - even if someone advertises incorrectly for a reason,then they obviously desire incorrect matches to occur.

The system checks whether the type (expressed through concepts in a OWL-S ontology) of each request output is matched by the advertisement outputs, and each advertisement input is matched by a request input. (Names are not checked.) Matches are ranked at four levels based on the relationship between the compared outputs or inputs: **exact** if both are the same, or if the request is an immediate subtype of the advertisement (the assumption that subtypes will retain the same set of outputs is required explicitly in [PKPS02]). **PlugIn** if the request is subsumed by the advertisement. **Subsumes** means that the request X is more general than the advertisement A (X subsumes A) so that it cannot be completely provided for. **Failure** means that "no subsumption relation between advertisement and request is identified".

The ranking shows two oddities. Notably, **PlugIn** is ranked lower than **exact** because there is an implicit assumption that if provider A is more general than B, A will most likely provide less complete coverage than B, e.g., that a service that offers the functionality *vehicle* is less likely to have all different models of *sedan* listed than one that merely advertises *sedan*. Clearly this assumption is strongly dependent on domain specific factors.

Also, **Failure** is determined as the absence of subsumption between advertisement and request (in either direction). Let us postulate a situation where provider A is subsumed by request X (matching 3 of 7 outputs), provider B matches 5 outputs, but provider B has an extraneous output not required by X. In this case, the match between X and B is a **failure** and A will be chosen as **Subsumes** despite the fact that B would have provided something closer to X and there would have been no harm to X from the extraneous output. We will see another case of this in the next example.

Li and Horrocks [LH03] are another example of OWL-S use for matchmaking. Their representation represents advertisement instances as special concepts with the argument that TBox level reasoning is more effective than ABox reasoning, but the criteria for effectiveness are not specified. Li and Horrocks have a slightly more involved list of matching rankings that is likewise based on subsumption only but includes a separate *intersection* ranking. However this is still not considered satisfactory and "excess" information is separated from the service profile to be moved into separate *providedBy* and *requestedBy* slots with the argument that "there is too much information and this makes it difficult to use automated reasoning techniques". Notably, there is no attempt to include precondition or effect specifications.

In [NM02], the semantics of Web services specified in OWL-S are expressed through Petri nets which are then used to simulate execution of the service and verify dynamic properties. There is no automatic composition.

3.3.1 OWL-S evaluation

Overall OWL-S (and OWL-S) can be found to have certain drawbacks concerning the matching process, not to speak of actual composition.

As pointed out in [vSSBR03], OWL-S can only express control patterns within one service. In [LH03], the restriction is assumed that individuals are not related to each other via properties.

When putting OWL-S to practical use (although still in an artificial environment) Sabou et al. found a number of other shortcomings [SRvS03]. These include the fact that profile and process descriptions invite inconsistency, that Profile preconditions and effects cannot refer to those of the corresponding process, that the specification syntax is repetitive, difficult, and therefore error-prone, and that OWL-S ignores standard software engineering concepts, thus preventing reuse and exploitation of parametric polymorphism in service descriptions. In [BLW04], other properties such as the inability to use variables in OWL expressions, are mentioned.

3.4 Web Service Composition as Configuration

In the configuration area, although configuring software has long been a topic of research [YMS02, HG02, ASM03], much of this work actually concentrates on software configuration management topics, and the properties and behaviours described tend to focus on parameter settings, module incompatibilities, and interaction with the execution environment more than on the actual behavior of the software. Web services provide a somewhat different view, in that they postulate, by definition, complete encapsulation of the execution environment combined with an exclusively execution oriented external description through their IOPE's. We are therefore interested in examining how well this information can be represented with the classical style of configuration descriptions as described, e.g., in [FFJ$^+$03].

We therefore assume the existence of a (logic-based or constraint-based) representation language for describing the domain constraints associated with a set of component types organized in a monotonic inheritance graph (which could be called a subsumption hierarchy except that full subsumption reasoning is usually not needed, but instead reasoning at the instance level is). Instances have scalar (numerical, string, or boolean) *attributes* (which may be subject to constraints) and connections to other components via *ports*. The specific attributes (and sometimes their predefined fixed value), ports, and constraints depend on the component's type.

We assume the definition of a configuration task $(DD, SRS, CLANG)$ as a triple of a domain description DD, a set of system requirements SRS, and the set of possible concepts $CLANG$ for the description of configuration solutions [FFJ$^+$03]. (Complexity of course depends on the expressiveness of the language used for the descriptions and we do not discuss it further here.)

3.5 Key component based model

First, it must be recognized that virtually all approaches defined above have merely pushed the UDDI approach down at most one level. Services may be matched in terms of IOPE's, but the inputs and outputs are generally assumed to be identically named (thus, it is assumed that there exists a common ontology for service parameter names although not necessarily for service names). Under these conditions the expression of the functionality of the service seems to devolve naturally down to the outputs which they produce.

Now let us consider the classical approach used by configuration tools with object-oriented representations. The classic approach is to express the functionality that one is looking for in terms of *key components* that in turn require the existence of other, auxiliary components until a finished configuration exists. Is it really appropriate to describe the key component of a matching process as, for example, a *car*? At this point it is useful to consider that what we are looking for is the specification of a *service* that looks for a car, and based on a particular set of inputs. It is that service specification that is the key component (and which will be used later to actually instantiate the service when the whole application is executed, so that we can actually refer to it as a template, or the description of a *call* to the service).

Therefore, we establish an inheritance hierarchy of service description types according to their inherent functionality and the attributes they possess. Service executions are instances of these. Consider the following example of a PC selling service from [LH03]:

- items are provided by an Actor with name Georgia ;
- items are PCs and the memory size is at least 128 Mb;
- the quantity of PCs being bought will be less than 200;
- the unit price is at least 700;
- the seller is guaranteed to have a creditLevel greater than 5;
- goods must be delivered before the 15/09/2002;
- goods must be delivered in Bristol

Using a notation for classes and constraints similar to the MML syntax from [FFJZ02] which is based on the OMG Object Constraint Language (OCL), we can describe this service template (contained in $CLANG$) as follows.[1] For brevity we omit the attribute definitions:

class CompSaleAd **subclass** ServiceProfile
...

[1] Alternately, a specification could simply be a set of constraints handed in together, but in terms of administration and storage, an actual specification entity seems more appropriate.

```
class Seller subclass Actor
...
seller1 = @Seller
        name = "Georgia";
        end;
class Sale1 subclass @CompSale
        seller = seller1;
        item = "PC";
        memory ≥ 128;
        processor = Set{Pentium3, Pentium4, Athlon...}
        deliveryLocation = "Bristol";
        inv quantity ≤ 200 end;
        inv unitPrice ≥ 700 end;
        inv seller.creditLevel > 5 end;
        inv deliveryDate < 15/09/2000
        end;
```

Assuming that the classes below are also in $CLANG$,

```
class ServiceRequest
        sv = @ServiceProfile
        end;
```

the request for a particular sale would appear as follows:

- the provider is an Actor with creditLevel greater than 5
- items are PCs and the Processor must be Pentium4;
- the unit price must be less than 700.

```
class CompOrderDescription
        subclass ServiceRequest end;
saleOrder = @SaleOrderDescription
        sv = @CompSaleAd;
        sv.item = "PC";
        sv.processor = "Pentium4;
        inv (sv.seller = @Actor)
           and (sv.seller.creditLevel > 5)
        inv sv.unitPrice < 700 end;
        end;
```

If we consider the above as a set of configuration requirements, then according to the definitions in [FFJ$^+$03], this order results in the creation of an instance of CompSale, or a subclass of it.

In summary, functionality is expressed through key components, Web service input restrictions and output requirements are described by constraints in the request. Properties of the Service are specified either as constraints or constants in the profile. Unsurprisingly, extraneous properties, unlike [LH03], do not prevent a match (and should not), while conditions on properties that the specific profile does not specify do prevent it (and should). Clearly, a match is better expressed in terms of consistency than mutual subsumption.

Flexible hierarchy matching. Above we have assumed the existence of a fixed inheritance hierarchy; in this services can still be selected not just by type but by checking their individual properties, in the spirit of resource-oriented configuration, which started out by using resources to express functionality of components [HJ91, FFJ$^+$03]. Providing some sort of automated subsumption computation for a set of classes is no problem (we note that [LH03] likewise assume that the concept hierarchy is not frequently changed).

Multiple service matching. Due to the instance level reasoning and the ability to use variables, it is possible to describe requests for multiple services, even of the same class, simply by specifiying multiple service variables in the request and adding constraints on them or their properties. This is particularly relevant in the context of coordination languages which may in fact specify particular combinations of services to be executed in particular patterns, sequentially, in parallel, or in mutual dependency. The ability to express this depends on the capability of the representation formalism, but is available in many approaches that provide an object-oriented component model underlying the representation (e.g., [Mai98, SFH98]).

The only other work that actually posits Web service composition in configuration terms is that of van Splunter et al. [vSSBR03] have addressed Web service composition using an agent-oriented approach. Their system, originally designed to reconfigure intelligent agents, was adapted to deal with OWL-S service descriptions. Its reasoning mechanisms are only vaguely described, consisting of separate tasks for reasoning about the design process (DPC), about requirements and their qualifications (RQSM), and about the design object description (DODM). As a result, the specific representation and reasoning mechanisms used are not clear, nor is their generic power. The authors admit to significant restrictions such as solely configuring linear service sequences.

3.5.1 System Architecture

The switch to a configuration reasoner actually simplifies the handling as no special purpose algorithms are needed. Of the five functionalities identified in [LH03], advertising a service and querying a service are handled using a standard constraint engine.

Above, we have not described any ranking scheme, and the consistency-based reasoning of the constraint solver merely distinguishes between matches and non-matches. However, switching to outcomes that rank matches can be done while staying within the constraint paradigm by utilizing an optimizing constraint solver [Tsa93].

Using a constraint language instead of using OWL-S directly is no great hindrance from the Web service perspective ([LH03] also diverges in using standard Description Logic (DL) syntax for writing their specifications.) Note that the constraint descriptions can be mapped to a description logic representation according to the guidelines set down in [FFJ 03], *as long as no constructs are used that exceed easy expressibility in DL's.* Unfortunately that is true of any sort of service description that involves multiple interacting services, but that is a consequence of the restricted DL syntax.

3.6 Planning

The ability to specify preconditions and effects (the "PE" in the "IOPE") is one of the greater expressive features in OWL-S. However, Sabou et al. [SRvS03] point out that their utility is currently restricted because of a disconnect between the description of Service Profiles and Processes. This issue deserves a closer look due to its effects on the computational characteristics.

The inclusion of pre- and postconditions, which is necessary for full service composition, suggests a situation where by specifying an initial and a goal condition, any set of services that provides the goal condition is composed. This transforms the problem from a "normal" configuration problem into a planning problem, and in fact numerous dedicated planning systems have been proposed to deal with the issue. In [PF02], a simple backward chaining planner is used, [SdF03] employs a classical STRIPS-like planning formalism, and [WPS 03] uses a Hierarchical Task Network planner optimised for effective task decomposition. In [TP04] a model-checking planner that can handle nondeterminism (such as the incorporation of services whose success is not known at the time of planning).

The key issue with all planning-based approaches to Web Service composition is scalability. In [TP04], a state of the art planner is used with an OWL-S representation that the authors describe as significantly more efficient than planning at the level of BPEL4WS descriptions. While small problems can be handled efficiently, a problem involving the matching of three processes with five parameters (with three values each) already reaches runtimes of several hours (Case 6, [TP04]). The current scalability of these methods is therefore questionable.

As examined in [MM03], the actual composition problem is significantly simplified if one assumes the existence of prespecified control flow between the services to be composed (as in the simple sequence of 2.1). This is the role taken by Web service cooperation languages that provide a framework (with full fledged control structures as in BPEL4WS) to anchor the individual steps (services) in an application. The pre- and postconditions are then constraints referring to the steps surrounding the current step in the cooperative plan that embeds the individual Web service calls, but the actual structure is fixed (in particular avoiding the problem of assembling complex control structures).

Notably, as expressed in [SRvS03, MM03], unless we are dealing with a fixed set of services, service descriptions are of limited use if there is no corresponding expanded UDDI interface that returns services that match requests. (Note that it can use the same description language as used for the original service requests and descriptions.)

3.7 Automata based methods

A final type of approach is the use of formal process models (in the spirit of process algebras or Communicating Sequential Processes) to describe service behaviors that can then be used in the composition task. E.g., [GHIS04] uses Mealy machines that describe permissible state changes within processes and the messages exchanged between them. Another approach [BGL 04] uses Deterministic Propositional Dynamic Logic as the underlying formalism to describe the automata's behavior. Like the "fixed ontology" approaches examined initially, these methods aim more at the orchestration problem - the connection of a specific set of given processes, rather than the general composition problem.

4 Conclusion

Web services are the up-and-coming concept for the implementation of distributed enterprise applications, for application integration, and interoperability standards. Yet the scope of the problem is so wide that even the boundary conditions for the service composition task are not clearly defined - from the orchestration approaches that aim at explicit procedural programming at BPEL4WS level using XML syntax, with basic services as the underlying statements, to the goal of automated, goal-directed programming, everything can currently be found as proposal status. Experimentation and acceptance by the audience will determine which approaches find widespread use.

Literatur

[ACKM04] Gustavo Alonso, Fabio Casati, Harumi Kuno und Vijay Machiraju. *Web Services*. Springer-Verlag, 2004.

[ASM03] T. Asikainen, Timo Soininen und Tomi Männistö. A Koala-Based Ontology for Configurable Software Product Families. In *IJCAI Workshop on Configuration*, Seiten 76–81, 2003.

[BGL 04] Daniela Berardi, Giuseppe De Giacomo, Maurizio Lenzerini, Massimo Mecella und Diego Calvanese. Synthesis of underspecified composite -services based on automated reasoning. In *Proc. ICSOC*, Seiten 105–114, 2004.

[BLW04] Steffen Balzer, Thorsten Liebig und Matthias Wagner. Pitfalls of OWL-S: a practical semantic web use case. In *Proc. ICSOC*, Seiten 289–298, New York, 2004. ACM Press.

[CCMW01] Erik Christensen, Francisco Curbera, Greg Meredith und Sanjiva Weerawarana. Web Services Description Language (WSDL) 1.1. Bericht, W3C, March 2001.

[ea02a] Anupriya Ankolekar et al. DAML-S: Web Service Description for the Semantic Web. In *Proc. ISWC*, Sardinia, Italy, Juni 2002.

[ea02b] T. Bellwood et al. UDDI Version 3.0. Bericht, UDDI.org, Juli 2002.

[FFJ$^+$03] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach, Markus Stumptner und Markus Zanker. Configuration Knowledge Representation for Semantic Web Applications. *AI EDAM*, 17(1), Januar 2003. Special issue on Configuration.

[FFJZ02] Alexander Felfernig, Gerhard Friedrich, Dietmar Jannach und Markus Zanker. Configuration Knowledge Representation Using UML/OCL. In *Proceedings UML 2002*, LNCS 2460. Springer, 2002.

[FLP$^+$03] Renato Fileto, Ling Liu, Calton Pu, Eduardo Delgado Assad und Claudia Bauzer Medeiros. POESIA: An ontological workflow approach for composing Web services in agriculture. *The VLDB Journal*, 12(1):352–367, 2003.

[GHIS04] Cagdas Evren Gerede, Richard Hull, Oscar H. Ibarra und Jianwen Su. Automated composition of e-services: lookaheads. In *Proc. ICSOC*, Seiten 252–262, New York, 2004. ACM Press.

[HG02] Lothar Hotz und Andreas Günter. Using Knowledge-Based configuration for Configuring Software? In *ECAI Workshop on Configuration*, Seiten 63–65, 2002.

[HJ91] M. Heinrich und E.W. Jüngst. A Resource-Based Paradigm for the Configuring of Technical Systems from Modular Components. In *Proceedings of the 7th IEEE Conference on AI Applications (CAIA)*, Seiten 257–264, 1991.

[KSR04] Rim Samia Kaabi, Carine Souveyet und Colette Rolland. Eliciting service composition in a goal driven manner. In *Proc. ICSOC*, Seiten 308–315, New York, 2004. ACM Press.

[LH03] Lei Li und Ian Horrocks. A software framework for matchmaking based on Semantic Web technology. In *Proceedings WWW 2003 Conference*, Seiten 331–339, Budapest, Mai 2003.

[Mai98] D. Mailharro. A classification- and constraint-based framework for configuration. *AI EDAM*, 12(4):383–397, 1998. Special issue on Configuration.

[MBE03] Brahim Medjahed, Athman Bouguettaya und Ahmed K. Elmagarmid. Composing Web Services on the Semantic Web. *The VLDB Journal*, 12(1):333–351, 2003.

[MM03] Daniel Mandell und Sheila A. McIlraith. Adapting BPEL4WS for the semantic Web. In *Proc. ISWC*, Sanibel Island, FL, Oktober 2003.

[NM02] Srini Narayanan und Sheila A. McIlraith. Simulation, verification and automated composition of web services. In *Proceedings WWW 2002 Conference*, Seiten 77–88. ACM Press, 2002.

[PF02] S. R. Ponnekanti und A. Fox. SWORD: A developer toolkit for Web service composition. In *Proceedings WWW 2002 Conference*, Honolulu, Mai 2002.

[PKPS02] Massimo Paolucci, Takahiro Kawamura, Terry Payne und Katia Sycara. Semantic Matching of Web Services Capabilities. In *Proc. ISWC*, Sardinia, Italy, Juni 2002.

[SdF03] Mithun Sheshagiri, Marie desJardins und Timothy Finin. A Planner for Composing Services Described in DAML-S. In *Workshop on Planning for Web Services, International Conference on Automated Planning and Scheduling*, Trento, Juli 2003.

[SFH98] Markus Stumptner, Gerhard Friedrich und Alois Haselböck. Generative Constraint-Based Configuration of Large Technical Systems. *AI EDAM*, 12(4), Dezember 1998.

[SRvS03] Marta Sabou, Debbie Richards und Sander van Splunter. An experience report on using DAML-S. In *Proceedings WWW 2003 Conference*, Budapest, August 2003.

[SS02] M. Schrefl und M. Stumptner. Behavior Consistent Specialization of Object Life Cycles. *ACM Transactions on Software Engineering and Methodology*, 11(1):92–148, 2002.

[Tea01] Business Process Team. ebXML Business Process Specification Schema. Bericht, ebXML.org, Mai 2001.

[The01] The DAML Services Coalition. DAML-S: Semantic Markup for Web Services. In *Proceedings of the International Semantic Web Working Symposium (SWWS)*, Stanford, Juli 2001.

[TP04] Paolo Traverso und Marco Pistore. Automated Composition of Semantic Web Services into Executable Processes. In *Proc. ISWC*, Seiten 380–394, 2004.

[Tsa93] Edward Tsang. *Foundations of Constraint Satisfaction*. Academic Press, 1993.

[vSSBR03] Sander van Splunter, Marta Sabou, Frances Brazier und Debbie Richards. Configuring Web Services using Structuring and Techniques from Agent Configuration. In *Proceedings 2003 IEEE/WIC Conference on Web Intelligence*, Beijing, Oktober 2003.

[WPS$^+$03] Dan Wu, Bijan Parsia, Evren Sirin, James Hendler und Dana Nau. Automating DAML-S Web services composition using SHOP2. In *Proc. ISWC*, Sanibel Island, FL, Oktober 2003.

[YMS02] Katariina Ylinen, Tomi Männistö und Timo Soininen. Configuring Software Products with Traditional Methods-Case Linux Familiar. In *ECAI Workshop on Configuration*, Seiten 5–10, 2002.

[YP02] Jian Yang und Mike P. Papazoglou. Web Component: A Substrate for Web Service Reuse and Composition. In *Proc. CAiSE 2002*, Toronto, 2002.