Computing service executions plans with probabilistic logics

Henrik Nottelmann and Norbert Fuhr

Institute of Informatics and Interactive Systems, University of Duisburg-Essen,
47048 Duisburg, Germany, {nottelmann,fuhr}@uni-duisburg.de

Abstract. Today, peer-to-peer services can comprise a large and growing number of services, e.g. search services or services dealing with heterogeneous schemas in the context of Digital Libraries. For a given task, the system has to determine suitable services and their processing order ("execution plan"). As peers can join or leave the network spontaneously, static execution plans are not sufficient. This paper proposes a logic-based approach for dynamically computing execution plans: Services are described in the DAML-S language. These descriptions are mapped onto probabilistic Datalog. Finally, logical rules are applied on the service description facts for determining matching services and deriving an optimum execution plan.

1 Introduction

Peer-to-peer architectures have emerged recently as an alternative to centralised architectures. The nodes in a peer-to-peer architecture provide services that can be used for improving information access to Digital Libraries. The most common services are search services, which provide retrieval in a Digital Library collection. Enhancing services retrieve additional, relevant documents (e.g. recommender systems), or modify the query prior to retrieval (e.g. query expansion). Mapping services transform queries and documents from one schema into another for bridging representational heterogeneity.

In peer-to-peer networks, nodes—and their services—can spontaneously join and leave, so they cannot be integrated in the system in a static way. Instead, the system (a so called match-making component) dynamically has to be compute an execution plan, a sequential order of services, for a given task (e.g. a user query). This execution plan can e.g. include schema mapping services if the schema of the query and the search service differ.

This paper proposes a logic-based approach for computing execution plans, which is based on the work presented in [4]:

1. DAML Services (DAML-S, [1]) is used for specifying service descriptions. DAML-S defines the vocabulary (an upper ontology) for describing arbitrary (also business-oriented) services. A lower ontology for Digital Library services is presented here.
2. In a next step, the DAML-S descriptions will be transformed into probabilistic logics. The resulting facts and logical rules can then be used by a match-making component for computing an execution plan.

3. Similar to resource selection in federated Digital Libraries, the match-making component should consider the costs of execution plans and compute an optimum selection. As this paper is focused on the first two topics (service descriptions, match-making), only a simple approach for considering service costs is presented. This approach has to be extended in future.

Other authors have proposed logic- or RDF-based approaches for finding suitable services before. In [6], services are modelled as processes (similar to DAML-S), and a simple query language for retrieving suitable processes is introduced. Furthermore, a first step towards partial matches (e.g. specialisation/generalisation) is made. [11] use RDF(S) advertisements for both producers and consumers, so match-making is reduced to RDF graph matching. A lisp-like logic is used by [7] for service capabilities descriptions. An AI planning component can infer an execution plan (sequence of services to call) for the given task.

In contrast, our approach combines DAML-S, the forthcoming standard for describing services in the Semantic Web, and probabilistic Datalog, a well-studied probabilistic extension to horn logics, in a natural way.

This paper is organised as follows: The next section gives a brief introduction into DAML Services. Section 3 extends DAML-S by a lower ontology for library services. These models will be transformed into probabilistic Datalog in Sec. 4. Match-making rules (see Sec. 5) can then be used for computing an optimum execution plan.

2 DAML Services (DAML-S)

DAML-S defines an upper ontology for describing services, expressed in DAML+OIL. Service descriptions consist of three different parts:

Profile: It describes what the services actually do, mainly by means of input and output parameters, preconditions and effects, and by the possibility of using different service types. The service profile will be used for match-making.

Process model: Processes describe how services work internally. They can be described either as atomic processes or as compositions of other services. Advanced match-making components can use the process model for an in-depth analysis.

Service grounding: The grounding can be used for calling the service, e.g. via WSDL. This implementation aspect is out of the scope of this paper.

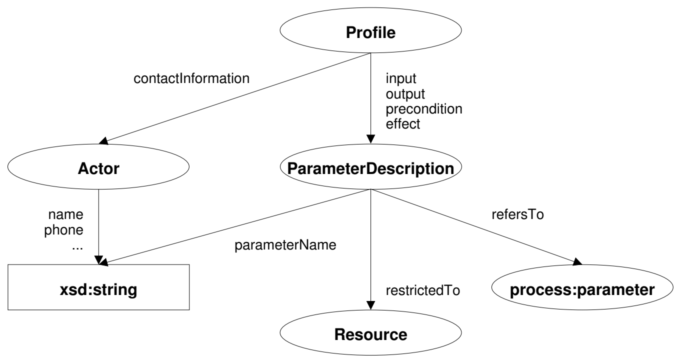
2.1 Service Profile

Every service has an associated profile. The profile (Fig. 1) gives a high-level description of the functionality of a service, and is intended to be used for match-making.

The contact information aims at developers who want to contact the responsible person (e.g. the system administrator) of the server, and can be neglected.

The parameter descriptions are more interesting. DAML-S supports four different kinds of service parameters: input parameters, output parameters, preconditions which have to be fulfilled in the physical world before the service can be executed, and effects the service has on the physical world.

Fig. 1. DAML-S profile definition



Preconditions and effects aim at E-Commerce applications. For a book selling service, the ordered book must be on stock, and after the service execution, the book will be delivered to the customer. In a Digital Library setting, preconditions and effects hardly play any role.

All parameters have a name, are restricted to a specific type (a DAML+OIL class or a XML Schema datatype), and refer to one parameter in the process model (see below).

2.2 Process Model

The process model (Fig. 2) gives a more detailed view on how the service actually works. It can be used by a match-making component for an in-depth analysis of the services.

Every process is described by input and output parameters, preconditions and effects. Profile parameter descriptions can correspond to these process parameters.

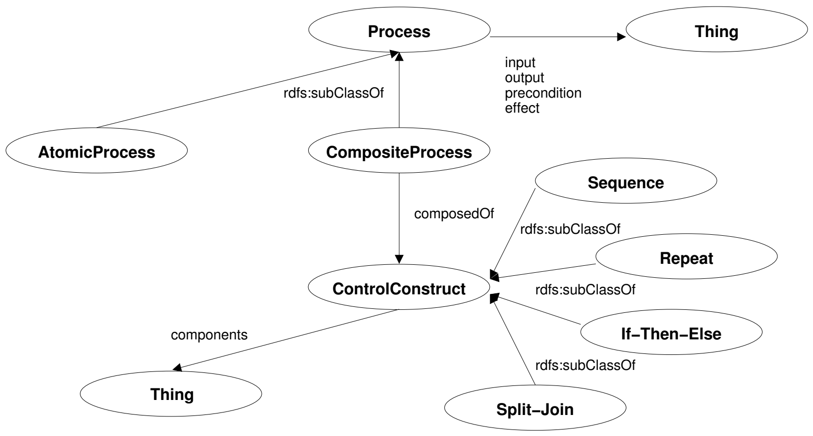
The definition of a parameter is shorter than in the profile. Each process parameter is a sub-property of the predefined `input`, `output` ... properties. No additional name (besides the property URI) is specified, in contrast to profile parameters.

DAML-S basically contains two types (as sub-classes) of processes: atomic and composite processes. Atomic processes are viewed as black boxes (like profiles).

Composite processes are defined as compositions of control constructs and other processes. Examples for control constructs are sequences of other control constructs (or processes), repetitions, conditions (if-then-else), or parallel execution of control constructs (or processes) with a synchronisation point at the end. Thus, composite processes allow for describing a service as a complex composition of other services. This is comparable to component-based software engineering, where often some scripting code is used to glue together existing components.

In a process model with only atomic processes, there is a one-to-one relationship with the corresponding profile. Thus, the atomic processes do not contribute any additional value to a service description. However, Sec. 5 shows that atomic processes are sufficient for match-making so far.

Fig. 2. DAML-S process definition



3 Lower ontology for library services

DAML-S only defines an upper service ontology, i.e. vocabulary for describing arbitrary services by means of their profiles, processes and process grounding. In addition, a domain-specific lower ontology is required. This lower ontology defines types of services (processes) which are used in the specific application area.

This section describes briefly a simple lower ontology for library services. As the parameter definition in the process model is simpler than in the profile (i.e., less DAML+OIL statements are required), atomic process descriptions are employed for match-making. Of course, all ideas and algorithms can also used together with profiles with only small changes.

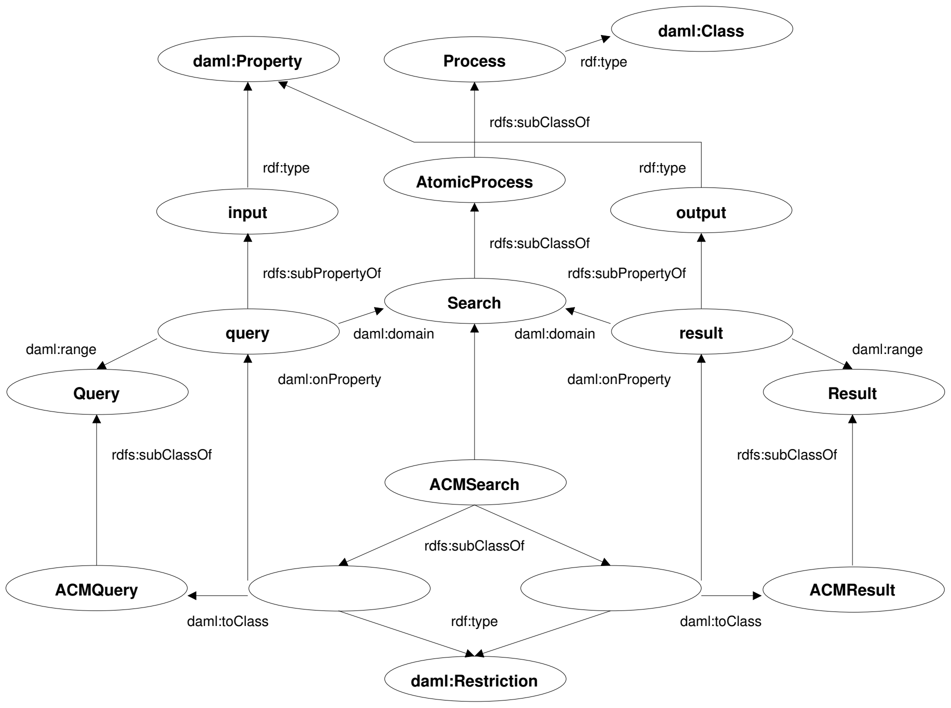
3.1 Search services

Search services are the most important services in distributed Digital Libraries. They receive a user query, retrieve useful documents from their associated collection, and return them to the caller.

A simple process model of search services is depicted in Fig. 3 (upper part). A search process is a special case (a DAML+OIL sub-class) of an atomic process. Every search process has exactly one query as input (the cardinality restrictions are omitted in the graph) and exactly one result (meant as a set of documents) as output. Thus, sub-properties of `input` and `output` are used. The ranges of these new properties are restricted to (generic) DAML+OIL classes `Query` and `Result`. They have to be defined somewhere else, but their exact definitions do not touch this discussion.

In a heterogeneous setting, search services probably use different schemas for expressing queries and representing documents. Typically some search services adhere

Fig. 3. Process model for search services



to Dublin Core (DC), e.g. from Open Archives. Other services might use specialised schemas, e.g. the ACM digital library, or services providing retrieval in art collections.

Thus, the description must also contain the schema the search service uses. This is modelled by creating schema-specific sub-classes for queries and results [8]. In the internal presentation, a library schema directly relates to a DAML+OIL "schema". For match-making, it is sufficient to consider the specific sub-types of queries and results.

The lower part in Fig. 3 shows the description of an ACM search service. Obviously, `ACMSearch` is a sub-class of the generic process class `Search`. The ranges of the `input` and `output` properties are restricted to ACM-specific query/result sub-classes.

With this extended description, a match-making component can clearly distinguish between search services using different schemas, and can plan accordingly.

3.2 Other library services

In large peer-to-peer-systems, and thus a large number of DLs to be federated, heterogeneity of DLs, especially w. r. t. the underlying schema, becomes a major issue. Each search service may use a different document structure. In federated Digital Libraries, e.g. MIND [8], users may query DLs in their preferred schema, and the system (i.e, schema mapping services) must perform the necessary transformations for each individual DL.

Query transformation services take one query referring to one specific schema as input and return the same query in another specific schema. In this paper, a service `DC2ACMQuery` is considered which transforms a DC query into an ACM query.

Similarly, a result transformation service like `ACM2DCResult` transforms a result (set of documents) adhering to one specific schema (here: ACM) into another schema (here: DC).

Finally, query modification services compute a new query for a given query based on some given relevance judgements, e.g. by applying a query expansion algorithm. Our scenario only contains one such service `DCQueryModification` working on DC queries and results.

4 DAM+OIL and Probabilistic Datalog

This section first introduces four-valued probabilistic Datalog. Then, it describes how DAML+OIL models, and thus DAML-S models, can be transformed into Datalog programs (facts for DAML+OIL statements, rules for expressing DAML+OIL semantics).

4.1 Four-valued probabilistic Datalog

Datalog [12] is a variant of predicate logic based on function-free Horn clauses. Negation is allowed, but its use is limited to achieve a correct and complete model. Rules have the form $h \leftarrow b_1 \wedge \cdots \wedge b_n$, where h (the "head") and b_i (the subgoals of the "body") denote literals[1] with variables and constants as arguments. A rule can be seen as a clause $\{h, \neg b_1, \ldots, \neg b_n\}$:

```
father(X,Y) :- parent(X,Y) & man(X).
```

This denotes that `father(x,y)` is true for two constants `x` and `y` if both `parent(x,y)` and `man(x)` are true. This rule has the head `father(X,Y)` and two body literals (considered as a conjunction) `parent(X,Y)` and `man(X)`. Negated literals start with an exclamation mark. Variables start with an uppercase character, constants with a lowercase character.

A fact is a rule with only constants in the head and an empty body:

```
man(peter). woman(mary).
```

Four-valued probabilistic Datalog (4vpD) [5, 3] uses an open-world assumption: if nothing is known for a fact, it has the truth value "unknown" (instead of "false" like in Datalog). On the other hand, a fact has the truth value "inconsistent" if there is evidence for both "true" and "false".

For facts and rules, probabilities for the events that the fact has the truth value "true"/"false"/"inconsistent" have to be specified manually; the probability for "unknown" can then be derived easily. The head of a rule can be negated (as a shorthand for $0/1/0$) for deriving negative knowledge:

[1] Literals in logics are different from literals in DAML+OIL!

```
0.6/0.2/0.2 man(jo).
!man(laura).
!man(O) :- woman(O).
```

Jo is a man with probability 0.6, not a man with probability 0.2, and the knowledge is inconsistent with probability 0.2. Laura is not a man, and women are not men.

If positive and negative rules are true for a fact in the head, then this fact is inconsistent.

4.2 Mapping DAML+OIL onto Datalog

DAML+OIL models and the DAML+OIL axiomatic semantics can be transformed into four-valued probabilistic Datalog programs [10]. For this, every property corresponds to a binary relation (where the property name equals the relation name). Then, DAML+OIL statements of the form "subject predicate object" can be mapped onto facts `predicate(subject,object)`.[2] If also URIs (optionally with a namespace name) are allowed for constants, the DAML+OIL model

```
<daml:Class rdf:id="dl:ACMSearch">
  <rdfs:subClassOf rdf:about="process:AtomicProcess"/>
</daml:Class>
```

can directly be transformed into with two deterministic facts:

```
type(dl:ACMSearch,daml:Class).
subClassOf(dl:ACMSearch,process:AtomicProcess).
```

The semantics of DAML+OIL modelling "primitives" can be modelled by rules:

```
type(O,C1)  :- subClassOf(C2,C1) & type(O,C2).
```

In some semantic constraints, e.g. the `disjointWith` property, negative knowledge has to be specified. Thus, it is a natural decision to switch to four-valued logics, as only here negative knowledge can be derived explicitly (using negated rule heads). Incorrect models can then be detected easily by checking for inconsistent facts:

```
!type(0,C1) :- disjointWith(C1,C2) & type(O,C2).
!type(0,C2) :- disjointWith(C1,C2) & type(O,C1).
```

5 Computing service execution plans

Our goal is to define pDatalog rules which can be used for computing an execution plan for a given task. These rules can then be applied directly on the facts which are generated from the DAML-S descriptions of the available services.

[2] In [10], a generic ternary relation `stat` is used as this allows generic rules for describing the DAML+OIL semantics. In this paper, however, the shorter version with binary relations is sufficient.

DAML-S service descriptions result in a relatively high number of facts. Ten facts are required for describing the generic search service (as shown above), and another ten facts for describing the ACM search service. This leads to rather long rules which are difficult to read for humans.

Thus, a new ternary auxiliary relation `service` is introduced, where the first argument contains the service name, the second one describes the type of the input parameter, and the last argument represents the output parameter type. If a service has more than one input or output value, the types are concatenated. Obviously, these facts can easily be derived from the existing knowledge.

```
service(dl:DCQueryModification,dl:DCQuery_DCResult,dl:DCQuery).
service(dl:DC2ACMQuery,dl:DCQuery,dl:ACMQuery).
service(dl:ACMSearch,dl:ACMQuery,dl:ACMResult).
service(dl:ACM2DCResult,dl:ACMResult,dl:DCResult).
```

Similar, our given task is defined by a ternary relation `task`:

```
task(mytask,dl:DCQuery_DCResult,dl:DCResult).
```

In a service chain, the output of a service must always be a super-set of the required input of the following service so that the latter one can be executed. A fact `match(T1,T2)` states that type `T1` is a super-set of the type `T2`:

```
match(dl:DCQuery_DCResult,dl:DCQuery).
match(dl:DCQuery_DCResult,dl:DCResult).
match(dl:ACMQuery,dl:ACMQuery).
...
```

5.1 Computing service chains

The basic idea is to determine a list of services which can be executed in sequential order (“service chain”). These chains then form potential execution plans.

Unfortunately, probabilistic Datalog does not allow for creating lists directly (like Prolog). Thus, service chains have to be defined recursively. The ternary relation `chain` encodes such a service chain. The first argument defines the service at the front of our chain, the third argument the service at the end of our chain. The second argument defines an arbitrary service somewhere in the middle. As a consequence, the chain

```
        DCQueryModification—DC2ACMQuery—ACMSearch—ACM2DCResult
```

is represented by two facts:

```
chain(dl:DCQueryModification,dl:DC2ACMQuery,dl:ACM2DCResult).
chain(dl:DCQueryModification,dl:ACMSearch,dl:ACM2DCResult).
```

Service chains with only two services are represented as:

```
chain(dl:ACMSearch,null,dl:ACM2DCResult).
```

Computation of service chains starts by chaining together two services with matching input and output types:

```
chain(S1,null,S2) :- service(S1,I1,O1) & service(S2,I2,O2) & match(O1,I2).

=> chain(dl:DCQueryModification,null,dl:DC2ACMQuery).
   chain(dl:DC2ACMQuery,null,dl:ACMSearch).
   chain(dl:ACMSearch,null,dl:ACM2DCResult).
```

Longer chains can be derived by computing the transitive closure of the `chain` relation:

```
chain(S1,S,S2) :- chain(S1,S11,S) & chain(S,S22,S2).

=> chain(dl:DCQueryModification,dl:DC2ACMQuery,dl:ACMSearch).
   chain(dl:DCQueryModification,dl:DC2ACMQuery,dl:ACM2DCResult).
   chain(dl:DCQueryModification,dl:ACMSearch,dl:ACM2DCResult).
   chain(dl:DC2ACMQuery,dl:ACMSearch,dl:ACM2DCResult).
```

If there are two chains where the last service of the first chain equals the first service of the second chain, then obviously the two chains can be connected.

5.2 Computing execution plans

Not every computed service chain is a suitable execution plan for the given task. The input and output of the chain and the task must match: The input type of task must be a super-set of the input type of the chain, and the output type of the chain must be a super-set of the output type of the task:

```
plan(T,S1,S,S2) :- task(T,TI,TO) & chain(S1,S,S2) &
                   service(S1,I,O1) & match(TI,I) &
                   service(S2,O2,O) & match(O,TO).

=> plan(dl:DCQueryModification,dl:DC2ACMQuery,dl:ACM2DCResult).
   plan(dl:DCQueryModification,dl:ACMSearch,dl:ACM2DCResult).
   plan(dl:DC2ACMQuery,dl:ACMSearch,dl:ACM2DCResult).
```

The complete execution plan (all services in the correct order) can be determined by iteratively traversing the `chain` relation. The fact database is queried for services between two services for which it is already known that they are in the plan.

The algorithm starts with the first and the intermediary service:

```
?- chain(dl:DCQueryModification,S,dl:ACMSearch).

=> (dl:DC2ACMQuery).
```

Thus, the DC2ACM query transformation service is in the plan between the query modification and the ACM search service.

It is still unclear if there are other services in that chain, so the procedure has to be repeated:

```
?- chain(dl:DCQueryModification,S,dl:DC2ACMQuery).

=> (null).
```

Thus, the query modification and the DC2ACM query transformation service have to be executed directly one after another.

This scheme has to be repeated until the complete execution plan is known.

5.3 Optimum execution plan selection

A match-making component has to choose one of the computed execution plans. In the context of search service selection, the concept of costs (combining e.g. time, money, quality) has been used for computing an optimum selection in the decision-theoretic framework [2]. Here, time and money ("effort") are separated from the number of relevant documents ("benefit"). The costs are later computed as the weighted difference between the effort and the benefit; the user-specific weights *ec* and *bc* allow for chosing different selection policies (e.g. good results, fast results).

If a service chain consists of two services, where each of them has its designated effort, then the effort of the service chain is the sum of the efforts of the two services. Its benefit has to be computed as the product of the benefits of the two services, as non-search services, e.g. query modification services, do not retrieve a fixed number of relevant documents. So, their benefit can be given relatively to the benefit of a search service only.

Typically, only distributions for the effort and benefit are given. Thus, two binary relations `effort` and `benefit` are used for specifying the distributions independently:

```
0.7 effort(dl:ACMSearch,10).  0.3 effort(dl:ACMSearch,12).
0.6 benefit(dl:ACMSearch,20). 0.4 benefit(dl:ACMSearch,30).
```

The relation `chain` is extended by two additional arguments for the effort and benefit of the whole service chains, `plan` is extended by one argument for the costs of the execution plan:

```
chain(S1,null,S2,E,B) :- service(S1,I1,O1) & effort(S1,E1) & benefit(S1,B1) &
                         service(S2,I2,O2) & effort(S2,E2) & benefit(S2,B2) &
                         match(O1,I2) & add(E,E1,E2) & mult(B,B1,B2).
chain(S1,S,S2,E,B) :-    chain(S1,S11,S,E1,B1) & chain(S,S22,S2,E2,B2) &
                         add(E,E1,E2) & mult(B,B1,B2).

plan(T,S1,S,S2,C) :- task(T,TI,TO) & chain(S1,S,S2,E,B) &
                     service(S1,I,O1,SE1,SB1) & match(TI,I) &
                     service(S2,O2,O,SE2,SB2) & match(O,TO) &
                     mult(SE,E,ec) & mult(SB,B,bc) & sub(C,SE,SB).
```

Expected costs can the be computed for every execution plan, and the execution plan with lowest expected costs has to be selected.

For simplicity, only exacts efforts and benefits are considered for the other services (in contrast to distributions):

```
effort(dl:DCQueryModification,4).    benefit(dl:DCQueryModification,1.2).
effort(dl:DC2ACMQuery,5).            benefit(dl:DC2ACMQuery,0.8).
effort(dl:ACM2DCResult,5).           benefit(dl:ACM2DCResult,0.8).
```

This yields these facts for the long chain (with four services):

```
=> 0.42 chain(dl:DCQueryModification,...,dl:ACM2DCResult,24,15.36).
   0.18 chain(dl:DCQueryModification,...,dl:ACM2DCResult,26,15.36).
   0.28 chain(dl:DCQueryModification,...,dl:ACM2DCResult,24,23.04).
   0.12 chain(dl:DCQueryModification,...,dl:ACM2DCResult,26,23.04).
```

The four facts for the shorter chain are similar.

With user-defined parameters *ec* and *bc*, the system generates eights probabilistic facts for executions plans (four for the short plan, the other four for the longer plan).

For $ec = 0.8$ and $bc = 0.2$ (preferring fast chains), the expected costs are 13.408 for the short plan and 15.9936 for the long plan. Thus, the short plan (with lower expected costs) is selected.

For $ec = 0.2$ and $bc = 0.8$ (high quality results), the expected costs are -8.168 for the short and -9.8256 for the long plan, so the long plan is selected for this user policy.

6 Conclusion and outlook

This paper presents a logic-based approach for computing execution plans. An execution plan in this scenario is a sequentially ordered list of services (“service chains”), e.g. search services, or schema mapping services. An execution plan is a correct service chain whose input and output parameter match the given task.

DAML Services (DAML-S) is used for describing the input and output parameters and the behaviour. DAML-S only defines a vocabulary for services descriptions, an upper ontology. Lower ontologies have to be defined for specific domains; this paper introduces a simple ontology for library services.

These DAML-S descriptions are then mapped onto probabilistic Datalog. This allows for using logical inference for computing execution plans. Uncertain rules also support cost-based selection of an optimum execution plan.

In future, a more detailed (and complete) lower ontology for library services is required. Several service types are missing yet, e.g. for mapping between different ontologies and/or free text, for mapping between different languages, for summarisation and for information extraction. Furthermore, a more detailed ontology should also employ composite processes as these allow a more sophisticated service selection. For search services, methods for estimating costs already have been investigated [9]; for non-search services, it is an open problem how retrieval quality can be estimated.

In a practical environment, it might be useful to describe the computed execution plan itself as a composite process in DAML-S. Then, existing software components can be used to execute the plan by employing the service groundings of the involved services.

Finally, the logic-based service selection approach has to be embedded in a peer-to-peer environment. In a hybrid network with a few hubs (or ultra-peers, super-nodes) and large number of service provider peers, services could be known to all hubs. Then, the hub which initially receives the query could perform this task directly. Otherwise, a peer-to-peer protocol has to be implemented for collecting service descriptions, e.g. via DHTs.

7 Acknowledgements

This work is supported by the DFG (grant BIB47 DOuv 02-01, project PEPPER).

References

[1] DAML-S Coalition. DAML-S 0.9 draft release. Technical report. `http://www.daml.org/services/daml-s/0.9/`.

[2] N. Fuhr. A decision-theoretic approach to database selection in networked IR. *ACM Transactions on Information Systems*, 17(3):229–249, 1999.

[3] N. Fuhr. Probabilistic Datalog: Implementing logical information retrieval for advanced applications. *Journal of the American Society for Information Science*, 51(2):95–110, 2000.

[4] N. Fuhr and C.-P. Klas. Combining RDF and agent-based architectures for semantic interoperability in digital libraries. In *Proceedings of the DELOS-Workshop on Interoperability in Digital Libraries*. DELOS-Network of Excellence on Digital Libraries, 2001.

[5] N. Fuhr and T. Rölleke. HySpirit – a probabilistic inference engine for hypermedia retrieval in large databases. In H.-J. Schek, F. Saltor, I. Ramos, and G. Alonso, editors, *Proceedings of the 6th International Conference on Extending Database Technology (EDBT), Valencia, Spain*, Lecture Notes in Computer Science, pages 24–38, Heidelberg et al., 1998. Springer.

[6] M. Klein and A. Bernstein. Searching for services on the semantic web using process ontologies. pages 431–446, 2001. `http://www.semanticweb.org/SWWS/program/full/paper2.pdf`.

[7] D. McDermott, M. Burstein, and D. Smith. Overcoming ontology mismatching in transactions with self-describing service agents. pages 285–302, 2001. `http://www.semanticweb.org/SWWS/program/full/paper39.pdf`.

[8] H. Nottelmann and N. Fuhr. Combining DAML+OIL, XSLT and probabilistic logics for uncertain schema mappings in MIND. In *European Conference on Digital Libraries (ECDL 2003)*, Heidelberg et al., 2003. Springer.

[9] H. Nottelmann and N. Fuhr. Evaluating different methods of estimating retrieval quality for resource selection. In J. Callan, G. Cormack, C. Clarke, D. Hawking, and A. Smeaton, editors, *Proceedings of the 26st Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, New York, 2003. ACM.

[10] H. Nottelmann and N. Fuhr. pDAML+OIL: A probabilistic extension to DAML+OIL based on probabilistic Datalog. (Submitted for publication), 2004.

[11] D. Trastour, C. Bartolini, and J. Gonzalez-Castillo. A semantic web approach to service description for matchmaking of services. pages 447–462, 2001. `http://www.semanticweb.org/SWWS/program/full/paper52.pdf`.

[12] J. D. Ullman. *Principles of Database and Knowledge-Base Systems*, volume I. Computer Science Press, Rockville (Md.), 1988.