A logic-based approach for computing service executions plans in peer-to-peer networks

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 Datalog. Finally, logical rules are applied on the service description facts for determining matching services and finding an optimum execution plan.

1 Introduction

Peer-to-peer architectures have emerged recently as an alternative to centralised architectures. In the beginning, they have been mainly used for simple applications like file sharing with only primitive retrieval capabilities. Nowadays, they are employed more and more for advanced IR applications.

In the scenario used in this paper, users search for documents in a peer-to-peer network (a "retrieval task"). A user issues a query to the network. The query is routed through the network, and—without further interaction with the user—documents are retrieved and sent back to the user. Documents are structured through schemas. Thus, queries are also stated against a schema, and the retrieval task defines the schema of user query and the schema of the result documents (requested by the user).

In contrast to other approaches, we assume a heterogeneous peer-to-peer network. Each peer can use its own schema for representing its documents. In addition, a peer can offer different services which are specialised in solving a specific problem, e.g. for bridging the heterogeneity (mediating between different schemas), or for improving information access to Digital Libraries. So, here we deal with a heterogeneous network of services which are offered by peers.

Nodes can spontaneously join and leave the peer-to-peer network, so they cannot be integrated in the system in a static way. Thus, a match-making component compares the (retrieval) task with all services which are available at that time, and computes an execution plan, a sequential order of services. This execution plan can include (besides search services) e.g. 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 picks up some ideas presented in [5]:

1. DAML Services (DAML-S, [2]) is the forthcoming standard for machine-readable service descriotions in the Semantic Web, and thus also employed in this approach. DAML-S defines the vocabulary (an upper ontology) for describing arbitrary (originally mostly business-oriented) services. A lower ontology for Digital Library services (i.e., the description of actual services) is presented in this paper.
2. In a next step, the DAML-S descriptions are transformed into Datalog, a predicate horn logic. Logical match-making rules can then be applied on the resulting facts 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. This paper presents an approach for cost-optimum service selection, based on probabilistic logics.

Other authors have proposed logic- or RDF-based approaches for finding suitable services before. In [6], services are modelled as processes using the MIT process Handbook ontology, providing similar modelling primitives as DAML-S (see Sec. 2), and introduces a simple query language for retrieving suitable processes. As this query language only uses the syntactic model, semantics-preserving query-mutation operators (using e.g. specialisation/generalisation) are introduced. In contrast, RDF(S) advertisements are used in [14] for both services and clients, so match-making is reduced to RDF graph matching. A lisp-like notation for logical constructs is used by [7] for both service capabilities descriptions and for the service request. An AI planning component can infer an execution plan by iteratively adding services which minimise the remaining effort.

In [12], the quality-of-service of a service execution plan is considered. Similar to the decision-theoretic framework for service selection selection ("resource selection") [10, 3] and the general service selection model presented in this paper, costs are associated with each execution plan, and a local optimisation algorithm is applied for finding the optimum execution plan. A user specifies a query w. r. t. virtual operations, for which then matching web services are found.

A similar approach is taken in [17]. Here, composite services, and thus execution plans, are modelled as state charts. Then, different quality criteria (e.g. monetary price, execution time, reliability, availability) are combined into an overall cost measure for an execution plan. As it is not feasible to consider every possible execution plan, linear programming is then employed for finding an optimum execution plan.

Edutella [8], a metadata infrastructure for the P2P network JXTA, combines RDF and Datalog. In contrast to [11], it does not work on an ontology level, and only maps RDF statements onto Datalog facts, without preserving the semantics of RDF modelling primitives. When the RDF model contains a DAML-S service description, the derived Datalog facts can be used for searching for services with known properties.

In contrast, the approach presented in this paper combines DAML-S, probabilistic Datalog, a probabilistic extension to predicate horn logic, and a decision-theoretic model for finding the cost-optimum execution plans in heterogeneous peer-to-peer networks.

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 a vocabulary for describing services (an upper ontology). The service model is expressed in DAML+OIL. E.g., DAML-S contains classes for processes and properties for defining their input and output types.

DAML-S, however, does not contain any description of actual services; they have to be defined in application-specific lower ontologies.

Service descriptions in DAML-S consist of three different parts:

Profile: It describes what the services actually do, mainly by means of input and output parameters, preconditions and effects. In addition, different service types can be used for categorisation. 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., WSDL descriptions can be included in the service groundings. Together with the service process mode, it can be used for actually invoking the service. 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 here.

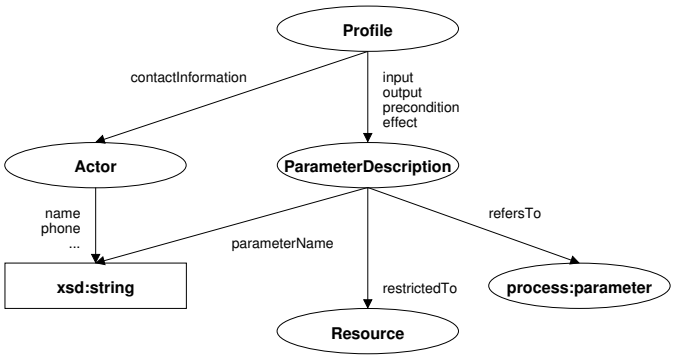
The parameter descriptions are more interesting. DAML-S supports four different kinds of service parameters: input parameters, output parameters, preconditions which have to be fulfilled in the physical world before the service can be executed, and effects the service has on the physical world.

Preconditions and effects mainly 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 the Digital Library setting used in this paper (pure retrieval task), preconditions and effects do not play any role. Thus, only inputs and outputs are used.[1]

Each parameter has a name (a string), is restricted to a specific type (a DAML+OIL class or an XML Schema data-type), and refers to one parameter in the process model (see below).

[1] This could easily be extended so that also preconditions and effects are considered.

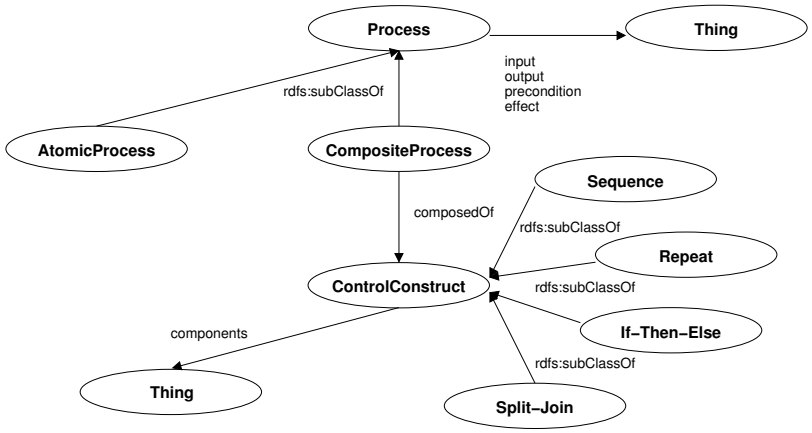
Fig. 1. DAML-S profile definition



2.2 Process Model

The process model (Fig. 2) gives a more detailed view on the service. As said before, it can be used by a match-making component for an in-depth analysis of the services.

Fig. 2. DAML-S process definition



Similar to the profile, a 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. The property URI is used for identification, no additional string is specified. In addition, each process parameter is a sub-property of the predefined `input`, `output` ... properties.

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 the usage of scripting code which glues together existing software components.

3 Lower ontology for library services

In addition to the DAML-S upper ontology, 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 briefly describes a simple lower ontology for library services. As the parameter definition in the process model is simpler than in the profile, atomic processes are employed here for the service descriptions.

3.1 Search services

Search services are among the 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 in the lower ontology, too, but left out as the exact definitions do not touch this discussion.

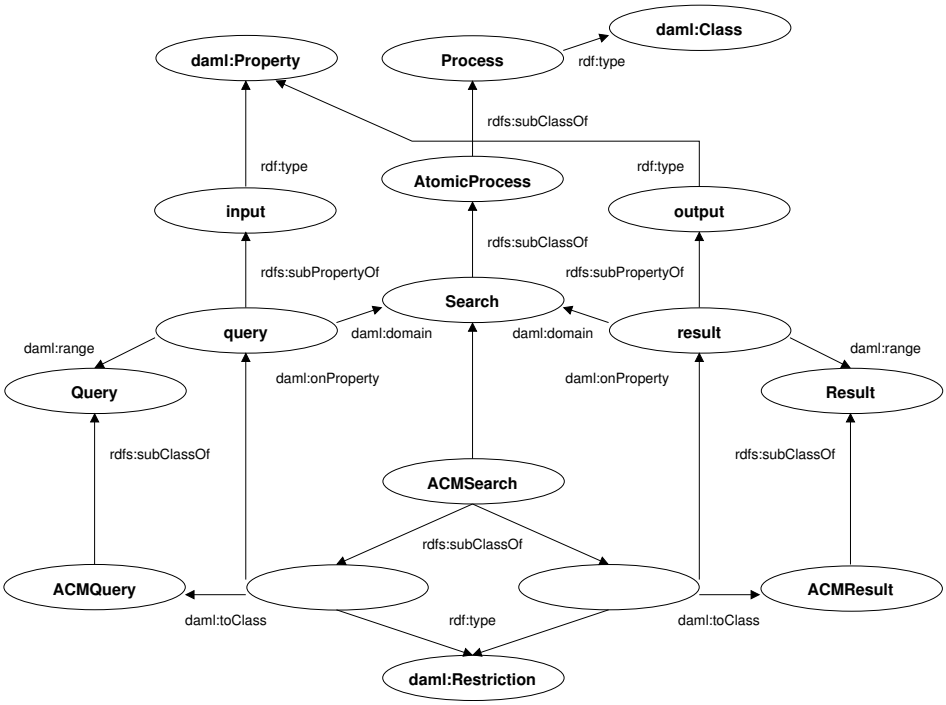
In a heterogeneous setting, search services probably use different schemas for expressing queries and representing documents. Typically some search services adhere to Dublin Core (DC), e.g. those operating on Open Archives data. Other services might use specialised schemas, e.g. the ACM digital library, 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 [9]. 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 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.

Fig. 3. Process model for search services



3.2 Other library services

In large peer-to-peer-systems, where a large number of DLs has 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 [9], 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.

Each schema mapping service mediates between exactly two different schemas ("input schema", "output schema"). If there is no schema mapping service available for a required mapping, then several schema mapping services have to be chained (with at least one intermediary schema).

There are two different kinds of schema mapping services:

– Query transformation services take a 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. This scenario only contains one such service `DCQueryModification` working on DC queries and results.

4 DAM+OIL and Datalog

This section first introduces deterministic and probabilistic Datalog. Then, it describes how DAML-S models are transformed into Datalog facts which can then be exploited by match-making rules.

4.1 Datalog

Datalog [15] 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 (see below). 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[2] 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) & male(X).
```

This denotes that `father(x,y)` is true for two constants `x` and `y` if both `parent(x,y)` and `male(x)` are true. This rule has the head `father(X,Y)` and two body literals (considered as a conjunction) `parent(X,Y)` and `male(X)`.

In addition, negated literals start with an exclamation mark. Variables start with an uppercase character, constants with a lowercase character. Thus the rule expresses that fathers are male parents.

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

```
parent(jo,mary).
```

The semantics are defined by well-founded models [16], which are based on the notion of the greatest unfounded set. Given a partial interpretation of a program, this is the maximum set of ground literals that can be assumed to be false.

Negation is allowed in Datalog as long as the program is modularly stratified [13] (in contrast to negation-as-failure in Prolog). Since the definition of modular stratification is rather complicated, we only give a simple explanation: In contrast to global stratification, modular stratification is formulated w.r.t. the instantiation of a program for its Herbrand universe. The program is modularly stratified if there is an assignment of ordinal levels to ground atoms such that whenever a ground atom appears negatively in the body of a rule, the ground atom in the head of that rule is of strictly higher level, and whenever a ground atom appears positively in the body of a rule, the ground atom in the head has at least that level.

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

4.2 Probabilistic Datalog

In probabilistic Datalog [4], every fact or rule has a probabilistic weight attached, prepended to the fact or rule:

```
0.5 male(X) :- person(X).
0.5 male(jo).
```

Semantics of pDatalog programs are defined as follows: The pDatalog program is modelled as a probability distribution over the set of all "possible worlds". A possible world is the well-founded model of a possible deterministic program, which is formed by the deterministic part of the program and a subset of the indeterministic part. As for deterministic Datalog, only modularly stratified programs are allowed.

Computation of the probabilities is based on the notion of event keys and event expressions, which allow for recognising duplicate or disjoint events when computing a probabilistic weight.

Facts and instantiated rules are basic events (identified by a unique event key). Each derived fact is associated with an event expression that is a Boolean combination of the event keys of the underlying basic events. E.g., the event expressions of the subgoals of a rule form a conjunction. If there are multiple rules for the same head, the event expressions corresponding to the rule bodies form a disjunction. By default, events are assumed to be independent, so the probabilities of events in a conjunction can be multiplied.

4.3 Transforming DAML-S models into Datalog

The services describes by DAML-S have to be transformed into a Datalog program which can be used for match-making.

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, the given task is defined by a ternary relation `task`:

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

As shown above, deterministic Datalog is sufficient for modelling services and tasks. Probabilistic Datalog will be employed later for computing optimum execution plans.

Alternatively, a generic mapping from any DAML+OIL model onto a Datalog program [11] could be employed.

5 Computing service execution plans

The goal is to define Datalog 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.

In a service chain, the output type of a service must be a super-set of the input type of the following service according to the following definition: Every single type in the input must be a sub-set of one of the types in the output.

More formally: Let the output type of a service be $OT := OT_1 \times OT_2 \times \cdots \times OT_k$ and the input type of another service by $IT := IT_1 \times IT_2 \times \cdots \times IT_l$. Then, OT and IT match if and only if for each $1 \leq i \leq l$ there is a $1 \leq j \leq k$ so that IT_j is a sub-set of OT_j, i.e. $IT_i \subseteq OT_j$.

In Datalog, this is encoded by facts `match(OT,IT)`:

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

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, 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, the third argument the service at the end of the chain. The second argument defines an arbitrary service somewhere in the middle (or equals `null`, if there is no other service).

As a consequence, the chain

$$\texttt{DCQueryModification} \longrightarrow \texttt{DC2ACMQuery} \longrightarrow \texttt{ACMSearch} \longrightarrow \texttt{ACM2DCResult}$$

can be represented by the following facts:

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

Computing service chains starts with finding chains of exactly 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:ACMSearch,dl:ACM2DCResult).
   plan(dl:DCQueryModification,dl:DC2ACMQuery,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 plan contains the DC2ACM query transformation service between the query modification and the ACM search service.

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

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

=> (null).


?- chain(dl:DC2ACMQuery,S,dl:ACMSearch).

=> (null).
```

Thus, the query modification and the DC2ACM query transformation service have to be executed directly one after another. The same holds for the query transformation and the search service.

Now, the second part of the chain has to be investigated. The result is that DC2ACMQuery and ACMSearch have to be executed without any service between them:

```
?- chain(dl:ACMSearch,S,dl:ACM2DCResult).

=> (null).
```

Thus, all complete execution plans can be determined based on the logic program.

5.3 Optimum execution plan selection

The match-making component might compute a large number of potential execution plans, and only one of them should be selected and executed.

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 [10, 3]. This framework gives a theoretical justification for selecting the best search services.

In this paper, a similar approach is applied to the more general problem of execution plan selection. Again, the notion of costs (of an execution plan) is used. For computation reasons, 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. User-specific weights *ec* and *bc* allow for choosing 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 must be specified relatively to the benefit of a search service.

Typically, not exact values, but 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).
```

For computing the costs of execution plans, 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).
```

When only distributions for the costs are given, the rules compute the distribution of the costs. These distributions can be used for computing expected costs for every execution plan (outside logics). Finally, the execution plan with lowest expected costs has to be selected.

In the example, only exacts efforts and benefits are considered for the other services (for simplicity):

```
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 (starting directly with the query transformation service) 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), these facts are computed for the long plan:

```
=> 0.42 plan(dl:DCQueryModification,...,dl:ACM2DCResult,16.13).
   0.18 plan(dl:DCQueryModification,...,dl:ACM2DCResult,17.73).
   0.28 plan(dl:DCQueryModification,...,dl:ACM2DCResult,14.59).
   0.12 plan(dl:DCQueryModification,...,dl:ACM2DCResult,16.19).
```

The expected costs for the long chain are computed by:

$$0.42 \cdot 16.13 + 0.18 \cdot 17.73 + 0.28 \cdot 14.59 + 0.12 \cdot 16.19 = 15.9936.$$

In contrast, the short plan has expected costs of 13.408. 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 in peer-to-peer networks offering heterogeneous services. 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.

The forth-coming standard for semantical service descriptions, DAML Services (DAML-S), is used for describing the input and output parameters and the behaviour. DAML-S employs DAML+OIL for defining a vocabulary (upper ontology) for describing services (e.g. as processes with input and output parameters). In this paper, we add a lower ontology for library services.

These DAML-S descriptions are mapped onto Datalog facts. This allows for using logical inference for computing execution plans. Uncertain facts and rules also support cost-based selection of an optimum execution plan. Instead of ternary predicate `service` (which describes the input and output types), a generic one-to-one mapping from DAML+OIL onto Datalog [11] could be employed.

In addition, the lower ontology for library services should be extended. 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, services could be described as composite processes for a more sophisticated service selection.

For search services, methods for estimating costs already have been investigated [10]; for non-search services, this still has to be investigated in detail.

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 will be embedded in a peer-to-peer environment. We will use a hierarchical network with a small number of hubs (also called ultra-peers, super-nodes) and large number of service provider peers. Hubs are computers with high computation capabilities and fast internet access, and will be responsible for computing execution plans. Either all services are known to all hubs, or 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] I. F. Cruz, S. Decker, J. Euzenat, and D. L. McGuinness, editors. *Proceedings of the International Semantic Web Working Symposium (SWWS)*, 2001.

[2] DAML-S Coalition. DAML-S 0.9 draft release. http://www.daml.org/services/daml-s/0.9/.

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

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

[5] 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.

[6] M. Klein and A. Bernstein. Searching for services on the semantic web using process ontologies. In Cruz et al. [1], pages 431–446. 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. In Cruz et al. [1], pages 285–302. http://www.semanticweb.org/SWWS/program/full/paper39.pdf.

[8] W. Nejdl, B. Wolf, S. Staab, and J. Tane. Edutella: Searching and annotating resources within an RDF-based P2P network. In *Proceedings of the Semantic Web Workshop, 11th Intl. World Wide Web Conf.*, 2002.

[9] 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.

[10] 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.

[11] H. Nottelmann and N. Fuhr. pDAML+OIL: A probabilistic extension to DAML+OIL based on probabilistic Datalog. In *ProceedingsInformation Processing and Management of Uncertainty in Knowledge-Based Systems*, 2004.

[12] M. Ouzzani. *Efficient Delivery of Web Services*. PhD thesis, Virgania Polytechnic Institute, USA, 2004. http://europa.nvc.cs.vt.edu/~mourad/mourad.ouzzani.pdf.

[13] K. Ross. Modular stratification and magic sets for Datalog programs with negation. *Journal of the ACM*, 41(6):1216–1266, Nov. 1994.

[14] D. Trastour, C. Bartolini, and J. Gonzalez-Castillo. A semantic web approach to service description for matchmaking of services. In Cruz et al. [1], pages 447–462. http://www.semanticweb.org/SWWS/program/full/paper52.pdf.

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

[16] A. van Gelder, K. Ross, and J. Schlipf. The well-founded semantics for general logic programs. *Journal of the ACM*, 38(3):620–650, July 1991.

[17] L. Zeng, B. Benatallah, M. Dumas, J. Kalagnanam, and Q. Z. Sheng. Quality driven web services composition. In *Proceedings of the 12th Intl. World Wide Web Conf.*, 2003.