Directory Services for Incremental Service Integration

Ion Constantinescu, Walter Binder, and Boi Faltings

Artificial Intelligence Laboratory
Swiss Federal Institute of Technology,
IN (Ecublens), CH–1015 Lausanne, Switzerland.
{ion.constantinescu,walter.binder,boi.faltings}@epfl.ch

Abstract. In an open environment populated by heterogeneous information services integration will be a major challenge. Even if the problem is similar to planning in some aspects, the number and the difference in specificity of services makes existing techniques not suitable and requires a different approach. Our solution is to incrementally solve integration problems by using an interplay between service discovery and integration alongside with a technique for composing specific partially matching services into more generic constructs. In this paper we present a directory system and a number of mechanisms designed to support incremental integration algorithms with partial matches for large numbers of service descriptions. We also report experiments on randomly generated composition problems that show that using partial matches can decrease the failure rate of the integration algorithm using only complete matches by up to **7 times** with no increase in the number of directory accesses required.[1]

1 Introduction

In a future service-oriented Internet service discovery, integration, and orchestration will be major building blocks. So far most research has been oriented towards service discovery and orchestration. Seminal work exists for service composition (e.g., [11] and [13]) which is now increasingly the focus of attention.

Earlier work on service integration has viewed service composition as a planning problem (e.g., [23]), where the service descriptions represent the planning operators, and applied traditional planning algorithms. Still the current and future state of affairs regarding web services will be quite different since due to the large number of services and to the loose coupling between service providers and consumers we expect that services will be indexed in directories. Consequently, planning algorithms will have to be adapted to a situation where operators are not known a priori, but have to be retrieved through queries to these directories.

Our approach to automated service composition is based on matching input and output parameters of services using type information in order to constrain the ways that services may be composed.

[1] The work presented in this paper was partly carried out in the framework of the EPFL Center for Global Computing and was supported by the Swiss National Science Foundation as part of the project MAGIC (FNRS-68155), as well as by the Swiss National Funding Agency OFES as part of the European projects KnowledgeWeb (FP6-507482) and DIP (FP6-507483).

J. Davies et al. (Eds.): ESWS 2004, LNCS 3053, pp. 254–268, 2004.

Consider the example (Fig. 1 (a)) of a personal agent that has the task of booking for its user tickets to a good movie and to find also directions for the user to reach the cinema, directions in the user's language. For this purpose the agent has to compose the following types of services: cinema services (show information and ticket booking), movie recommendation services, payment services, and geographical information services (GIS).

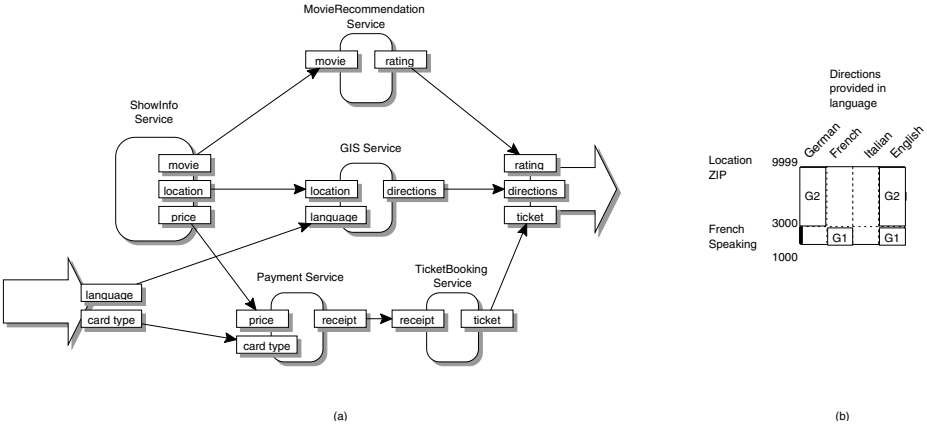

Fig. 1. Composition example: going out for a movie.

However, frequently a certain service cannot be applied because it requires a parameter to be within a certain range that is smaller than that of the possible inputs. Existing service composition algorithms will not be able to make use of such a service, even though a collection of distinct services may be able to handle the full range of the available inputs.

In the evening organizer example this could be the case of the GIS service. We presume a domain for Swiss GIS services which defines zip codes between 1000–9999 and is able to give directions in german, french, italian, and english language. In our setup we assume the existence of two GIS providers: G1 that is able to give directions in french and english for the french speaking part of Switzerland and G2 that is able to give directions in german and english for the rest. In this case requests for directions in english could be handled by a single service only if the zip codes are restricted to particular ranges: either between 1000 and 2999 (for G1) or between 3000 and 9999 (for G2).

Our approach (see Section 3) allows for *partially matching* types and handles them by computing and introducing software switches in the integration plan. In our example (Fig. 1 (b)) this could be done by composing together G1 and G2 using a software switch that will decide based on the runtime value of the zip code input parameter which of the two services to use. The value of composing partially matching services is evident as the composed service will work in more cases than any of G1 or G2. This is confirmed also by experimental results carried for two domains (see Section 5) which

show that using *partial matches* allows to decrease the failure rate of the integration algorithm that uses only *complete matches* by up to **7 times**.

We have developed a directory service with specific features to ease service composition. Queries may not only search for complete matches, but may also retrieve partially matching directory entries. As the number of (partially) matching entries may be large, the directory supports incremental retrieval of the results of a query. This is achieved through sessions, during which a client issues queries and retrieves the results in chunks of limited size. Sessions are well isolated from each other, also concurrent modifications of the directory (i.e., new service registrations, updates, and removal of services from the directory) do not affect the sequence of results after a query has been issued. We have implemented a simple but very efficient concurrency control scheme which may delay the visibility of directory updates but does not require any synchronization activities within sessions. Hence, our concurrency control mechanism has no negative impacts on the scalability with respect to the number of concurrent sessions.

Our contributions are twofold: first we show how service descriptions can be numerically encoded and how indexing techniques can be used in order to create efficient service directories that have millisecond response time for thousands of service descriptions. As our second contribution we describe functionality of the directory specific for solving service composition problems like the identification of partially matching service descriptions (e.g., see [6]), sessions for ensuring consistent directory views to the integration algorithm, and finally concurrency control for allowing simultaneous operations on the directory structure by different clients.

This paper is structured as follows: next in Section 2 we show how service descriptions can be numerically encoded as sets of intervals and we described in more details the assumed semantics for service descriptions and integration queries. In Section 3 we present how an index structure for multidimensional data can be exploited such that numerically encoded service descriptions are efficiently retrieved using queries for complete and partial type matches. In Section 4 we introduce some extensions of the directory system specific to service composition that enable consistent views of the directory for the client during the integration process, the retrieval at each integration step of only services that are unknown and concurrency control between different clients. In Section 5 we present some experimental results on randomly generated problems. In Section 6 we review some existing directory systems. Finally, Section 7 concludes the paper.

2 Numerically Encoding Services and Queries

Service descriptions are a key element for service discovery and service composition and should enable automated interactions between applications. Currently, different overlapping formalisms are proposed (e.g., [21], [19], [7], [8]) and any single choice could be quite controversial due to the trade-off between expressiveness and tractability specific to any of the aforementioned formalisms.

In this paper, we will partially build on existing developments, such as [21], [1], and [7], by considering a simple table-based formalism where each service is described through a set of tuples mapping service parameters (unique names of inputs or outputs)

to parameter types (the spaces of possible values for a given parameter). Parameter types can be expressed either as sets of intervals of basic data types (e.g., date/time, integers, floating-points) or as classes of individuals.

Class parameter types can be defined through a descriptive language like XML Schema [22] or the Ontology Web Language [20]. From the descriptions we can then derive either directly or by using a description logic classifier a directed graph (DG) of simple is-a relations (e.g., the is-a directed acyclic graph (DAG) for types of cuisine in Fig. 2 (a) derived from the ontology above).

For efficiency reasons, we represent the DG numerically. We assume that each class will be represented as a set of intervals. Then we encode each parent-child relation by sub-dividing each of the intervals of the parent (e.g., in Fig. 2 (b) Mediteraneean is sub-divided for encoding Italian and French cuisine); in the case of multiple parents the child class will then be represented by the union of the sub-intervals resulting from the encoding of each of the parent-child relations (e.g., the FrancoAsianFusion in Fig. 2 is represented through sub-intervals of the Asian and French concepts).

Since for a given domain we can have several parameters represented by intervals, the space of all possible parameter values can be represented as a rectangular hyper-space, with a dimension for each parameter.

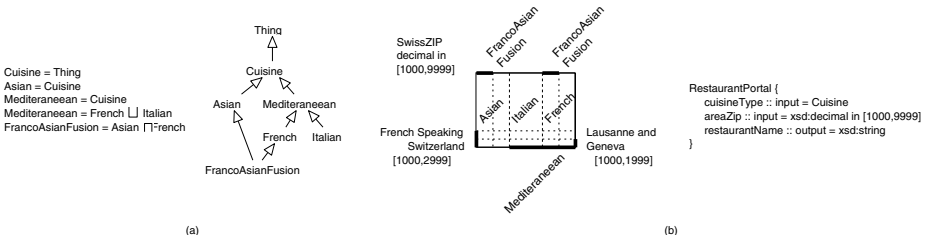

Fig. 2. An example domain: a restaurant recommendation portal.

Let's consider as an example (see Fig. 2 (b)) a restaurant recommendation portal that takes the user preference for a cuisine type and the Swiss zip-code (four digit numbers between 1000 and 9999) of the area of interest and will return a string containing a recommended restaurant located in a given area.

In this example the service will accept for the cuisineType parameter any of the keywords $Mediteraneean$, $Asian$, $French$, $Italian$, or $FrancoAsionFusion$ and for the $areaZip$ any decimal between 1000 and 9999 representing the location of the area of interest.

We assume that input and output parameters of service descriptions have the following semantics:

– In order for the service to be invokable, a value must be known for each of the service input parameters and it has to be consistent with the respective parameter type. For primitive data types the invocation value must be in the range of allowed values

or in the case of classes the invocation value must be subsumed by the parameter type.
- Upon successful invocation the service will provide a value for each of the output parameters and each of these values will be consistent with the respective parameter type.

Service composition queries are represented in a similar manner but have different semantics:

- The query inputs are the parameters available to the integration (e.g., provided by the user). Each of these input parameters may be either a concrete value of a given type, or just the type information. In the second case the integration solution has to be able to handle all the possible values for the given input parameter type.
- The query outputs are the parameters that a successful integration must provide and the parameter types define what ranges of values can be handled. The integration solution must be able to provide a value for each of the parameters in the problem output and the value must be in the range defined by the respective problem output parameter type.

3 Service Composition with Directories

As one of our previous assumptions is that a large number of services will be available, the integration process has to be able to discover relevant services incrementally through queries to the service directory. Interleaving the integration process with directory discovery is another novelty of our approach.

Our composition algorithm builds on forward chaining, a technique well known for planning [3] and more recently for service integration [18]. Most previous work on service composition has required an explicit specification of the control flow between basic services in order to provide value-added services.

For instance, in the eFlow system [5], a composite service is modelled as a graph that defines the order of execution of different processes. The Self-Serv framework [2] uses a subset of statecharts to describe the control flow within a composite service. The Business Process Execution Language for Web Services (BPEL4WS) [4] addresses compositions where the control flow of the process and the bindings between services are known in advance. In service composition using Golog [14], logical inferencing techniques are applied to pre-defined plan templates.

More recently, planning techniques have been applied to the service integration problem [16, 23, 24]. Such an approach does not require a pre-defined process model of the composite service, but returns a possible control flow as a result.

Other approaches to planning, such as planning as model checking [9], are being considered for web service composition and would allow more complex constructions such as loops.

Informally, the idea of composition with forward chaining is to iteratively apply a possible service S to a set of input parameters provided by a query Q (i.e., all inputs required by S have to be available). If applying S does not solve the problem (i.e., still not all the outputs required by the query Q are available) then a new query Q' can be

computed from Q and S and the whole process is iterated. This part of our framework corresponds to the planning techniques currently used for service composition [18].

We will make the following assumptions regarding the service composition process:

- any service input or output in the composition will be assigned exactly *one semantic description* (through the name of the parameter) at the moment of its introduction (when provided either as a problem input or as a service output). Also we assume that the semantic definition of the parameter will not change for the rest of the composition independently of other services applied later to the parameter. Please note that the application of subsequent services could still cast the parameter to other different parameter types.
- during integration the view that the algorithm has of the directory will not change. This is similar to the isolation property of transactional systems.

3.1 Directories of Web Services

Currently UDDI is the state of the art for directories of web services (see Section 6 for an overview of other service directory systems). The standard is clear in terms of data models and query API, but suffers from the fact that it considers service descriptions to be completely opaque.

A more complex method for discovering relevant services from a directory of advertisements is matchmaking. In this case the directory query (requested capabilities) is formulated in the form of a service description template that presents all the features of interest. This template is then compared with all the entries in the directory and the results that have features compatible with the features of interest are returned. A good amount of work exists in the area of matchmaking including LARKS [17], and the newer efforts geared towards DAML-S [15]. Other approaches include the Ariadne mediator [11].

3.2 Match Types

We consider four match relations between a query Q and a service S, (see example for inputs in Fig. 3).

- **Exact** - S is an exact match of Q.
- **PlugIn** - S is a plug-in match for Q, if S could be always used instead of Q.
- **Subsumes** - Q contains S. In this case Q could be used under the condition that it satisfies some additional constraints such that it is specific enough for S.
- **Overlap** - Q and S have a given intersection. In this case, runtime constraints both over Q and S have to be taken into account.

It has to be noticed that the following implications hold for any match between query and service descriptions Q and S: $\mathbf{Exact}(\mathbf{Q},\mathbf{S}) \Rightarrow \mathbf{PlugIn}(\mathbf{Q},\mathbf{S})$ and $\mathbf{Exact}(\mathbf{Q},\mathbf{S}) \vee \mathbf{PlugIn}(\mathbf{Q},\mathbf{S}) \vee \mathbf{Subsumes}(\mathbf{Q},\mathbf{S}) \Rightarrow \mathbf{Overlap}(\mathbf{Q},\mathbf{S})$. Given also that a **Subsumes** match requires the specification of supplementary constraints we can

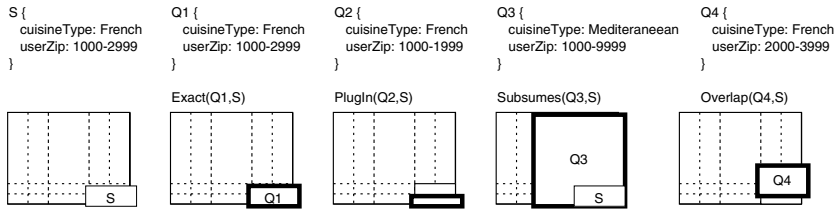

Fig. 3. Match types of inputs of query Q and service S by "precision": **Exact**, **PlugIn**, **Subsumes**, **Overlap**.

order the types of match by "precision" as following: **Exact**, **PlugIn**, **Subsumes**, **Overlap**. We consider **Subsumes** and **Overlap** as "partial" matches. The first three relations have been previously identified by Paolucci in [15].

Determining one match relation between a query description and a service description requires that subsequent relations are determined between all the inputs of the query Q and service S and between the outputs of the service S and query Q (note the reversed order of query and services in the match for outputs). Our approach is more complex than the one of Paolluci in that we take also into account the relations between the properties that introduce different inputs or outputs (equivalent to parameter names). This is important for disambiguating services with equivalent signatures (e.g., we can disambiguate two services that have two string outputs by knowing the names of the respective parameters).

In the example below, we show how the match relation is determined between the inputs available from the queries Q1, Q2, Q3, Q4 and the inputs required by service S.

3.3 Multidimensional Access Methods - GiST

The need for efficient discovery and matchmaking leads to a need for search structures and indexes for directories. We consider numerically encoded service descriptions as multidimensional data and we use in the directory techniques related to the indexing of such kind of information. This approach leads to local response times in the order of milliseconds for directories containing tens of thousands (10^4) of service descriptions.

The indexing technique that we use is based on the Generalised Search Tree (GiST), proposed as a unifying framework by Hellerstein [10]. The design principle of GiST arises from the observation that search trees used in databases are balanced trees with a high fanout in which the internal nodes are used as a directory and the leaf nodes point to the actual data. Each internal node holds a key in the form of a predicate P and can hold at the most a predetermined number of pointers to other nodes (usually a function of system and hardware constraints, e.g., filesystem page size). To search for records that satisfy a query predicate Q, the paths of the tree that have keys P that satisfy Q are followed.

4 Service Integration Sessions and Concurrency Control

As directory queries may retrieve large numbers of matching entries (especially when partial matches are taken into consideration), it is important to support incremental access to the results of a query in order to avoid wasting network bandwidth. Our directory service offers *sessions* which allow a user to issue a query to the directory and to retrieve the results one by one (or in chunks of limited size).

The session guarantees a consistent view of the result set, i.e., the directory structure and contents as seen by a session does not change. Concurrent updates (service registration, update, and removal) do not affect the sequence of query results returned within a session; sessions are isolated from concurrent modifications.

Previous research work has addressed concurrency control in generalised search trees [12]. However, these concurrency control mechanisms only synchronize individual operations in the tree, whereas our directory supports long-lasting sessions during which certain parts of the tree structure must not be altered. This implies that insertion and deletion operations may not be performed concurrently with query sessions, as these operations may significantly change the structure of the tree (splitting or joining of nodes, balancing the tree, etc.).

The following assumptions underly the design of our concurrency control mechanism:

1. Read accesses (i.e., queries within sessions and the incremental retrieval of the results) will be much more frequent than updates.
2. High concurrency for read accesses (e.g., 10^4 concurrent sessions).
3. Read accesses shall not be delayed.
4. Updates may become visible with a significant delayed, but feedback concerning the update (success/failure) shall be returned immediately.
5. The duration of a session may be limited (timeout).

A simple solution would be to create a private copy of the result set of each query. However, as we want to support a high number of concurrent sessions, such an approach is inefficient, because it wastes memory and processing resources to copy the relevant data. Hence, such a solution may not scale well and is not in accord with our first two assumptions.

Another solution would be to support transactions. However, this seems to be too heavy weight. The directory shall be optimized for the common case, i.e., for read accesses (first assumption). This distinguishes directory services from general-purpose databases.

Concurrency protocols based on locking techniques are not in accord with these assumptions either. Because of assumption 3, sessions shall not have to wait for concurrent updates.

In order to meet the assumptions above, we have designed a mechanism which guarantees that sessions operate on read-only data structures that are not subject to changes. In our approach the in-memory structure of the directory tree (i.e., the directory index) is replicated up to 3 times, while the actual service descriptions are shared between the replicated trees.

When the directory service is started, the persistent representation of the directory tree is loaded into memory. This master copy of the directory tree is always kept up to date, i.e., updates are immediately applied to that master copy and are made persistent, too. Upon start of the directory service, a read-only copy of the in-memory master copy is allocated. Session operate only on this read-only copy. Hence, session management is trivial, there is no synchronization needed. Periodically, the master copy is duplicated to create a new read-only copy. Afterwards, new sessions are redirected to the new read-only copy. The old read-only copy is freed when the last session operating on it completes (either by an explicit session termination by the client or by a timeout).

We require the session timeout to be smaller than the update frequency of the read-only copy (the duplication frequency of the master copy). This condition ensures that there will be at most 3 copies of the in-memory representation of the directory at the same time: The master where updates are immediately applied (but which is not yet visible to sessions), as well as the previous 2 read-only copies used for sessions. when a new read-only copy is created, the old copy will remain active until the last session operating on it terminates; this time span is bounded by the session timeout.

In our approach only updates to the master copy are synchronized. Updates are immediately applied to the master copy (yielding immediate feedback to the client requesting an update). Only during copying the directory is blocked for further updates. In accord with the third assumption, the creation of sessions requires no synchronization.

5 Evaluation and Assessment

This section first presents two domains used by our service integration testbed: one more concrete and another that provides more symmetry. A discussion of the results concludes this section.

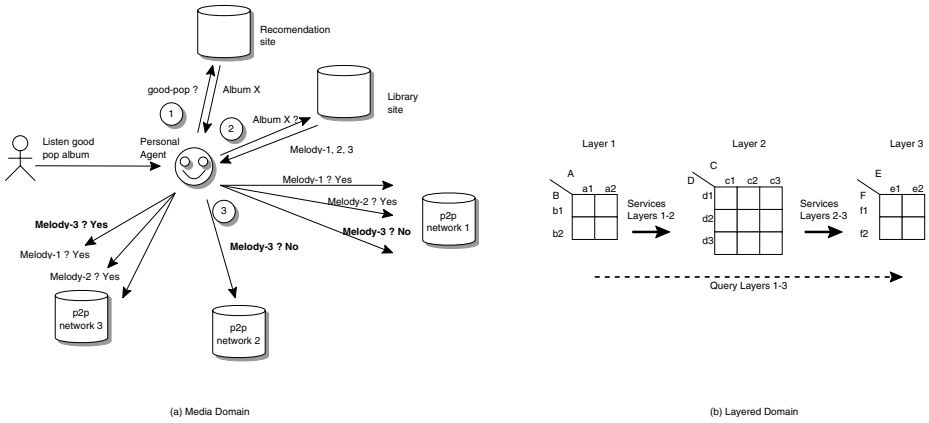

Fig. 4. Testbed domains.

5.1 The Media Domain

For experimental purposes we have considered the following scenario (see Fig. 4 (a)): a personal agent (PA) is delegated the task to find a music provider for an album of good pop music. Since its user doesn't like to listen only to spare melodies we assume all-or-nothing semantics: only if all the melodies of a given album can be found then this is a solution. Otherwise another album has to be selected.

First the PA has to determine what is currently considered as good pop and uses a recommendation site (e.g., billboard.com) to find Album X as a first option. Then the PA has to use a library site (e.g., cdcovers.cc) to determine which are the melodies in Album X - let's presume that they are Melody-1, Melody-2 and Melody-3. Then finally the PA searches different p2p networks for providers that have different melodies from the album.

In this context partial type matches are used for selecting peers that can provide (cover) a sub-set of the melodies of an album. Then by computing the software switch we select those peers that together can provide all the melodies from the album.

Since for solving the integration we need at least a recommendation, an album description and a provider for the melodies the minimum number of required directory accesses for any integration in this domain is 3.

5.2 The Layered Domain

We have defined also a more abstract domain (see Fig. 4 (b)) where we consider a number of layers that define sets of parameter names. Services are defined as transformations between parameters in adjacent layers and problems are defined between parameters of the first and last layer. For example, a possible service between layers1-2 with the parameters A, B could have as input the types A=a1, B=b1,b2 and for the output parameters C and D could have as types C=c2, c3 and D=d1, d2. A query could for the input parameters A, B the types A=a1,a2, B=b2 and for the output parameters E,F the type E=e1, F=f2.

For the purpose of our experiments, we have created environments with 3 layers for which the minimum number of required directory accesses for any integration is 2. Each of the layers defines two parameters with 11 possible subtypes for layers 1 and 3 and 63 possible overlapping subtypes for layer 2; between each two layers (1-2 or 2-3) there are 480249 possible services.

It has to be noted that in contrast with the media domain there is a symmetry regarding inputs and outputs in the configuration used for the layered domain.

5.3 Evaluation of Experimental Results

For both domains, we have randomly generated services and problems. For each specific type of services (e.g., album recommendation, album description, melody provider or service layer1-2, service layer2-3) or for queries (e.g., find good pop album in mp3 format or find a service that transforms an input in layer1 to an output in layer3) we had a pre-determined set of parameters. For actually generating the descriptions (service or

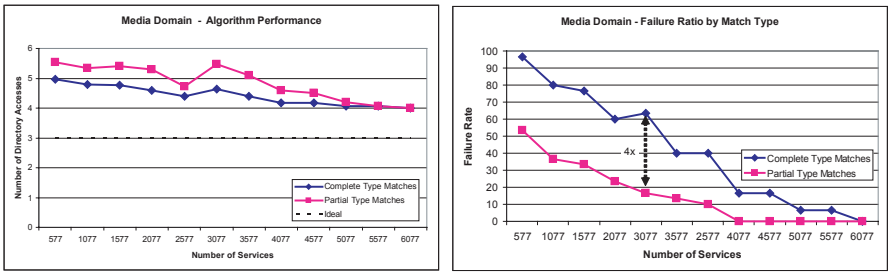

Fig. 5. The media domain.

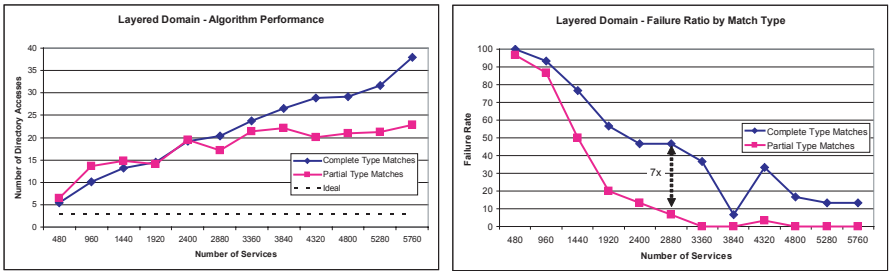

Fig. 6. The layered domain.

query) we randomly picked for any of the pre-determined parameters a random sub-type from a set of possible types.

We then solved the queries using first an algorithm that handles only *complete* type matches and then an algorithm that handles *partial* type matches (and obviously includes *complete* matches). We have measured the number of directory accesses and the failure ratio of the integration algorithms.

Fig. 5 (a) and Fig. 6 (a) show the average number of directory accesses for the algorithm using *complete* type matching versus the average number of directory accesses for the algorithm also using *partial* type matching.

Regarding performance, the first conclusion is that both algorithms scale well, as there is at most a slow increase in the number of directory accesses as the number of services in the directory grows. For the media domain the number of accesses actually decreases.

As expected, the algorithm using partial matches performs comparable with the one using complete matches. As it results from the experimental data for both domains the overhead induced by the usage of partial matches is not very significant and decreases as the directory gets saturated with services. This is probably due to the fact that having more choices makes the coverage problem intrinsic to the partial algorithm easier. More than that in the layered domain from some point the partial algorithm even performs better that the complete one (Fig. 6 (a) after 3000 services).

The most important result concerns the number of extra problems that can be solved by using partial matches and can be seen in Fig. 5 (b) and Fig. 6 (b). The graph show that the failure rate in the case of using only *complete matches is much bigger than the failure rate when* partial matches are used: **4 times** in the case of the Media domain and up to **7 times** in the case of the Layered domain. This shows that using partial matches opens the door for solving many problems that were unsolvable by the complete type matching algorithm.

6 Background and Related Work

In this section we will review a number of existing directory systems. As we will see next only some of the presented systems fullfill particular requirements of service integration like incremental access (UDDI, LDAP, CosTrader) and partial matches (LDAP, CoSTrader) but there is none that fulfills requirements that are more specific to service integration like the retrieval of only unkown services or maintaining the same view during the integration episode, issues that we address in our system trough the session mechanism.

6.1 UDDI

The Universal Description, Discovery and Integration (UDDI [19]) is an industrial effort to create an open specification for directories of service descriptions. It builds on existing technology standardized by the World Wide Web Consortium (http://www.w3c.org) like the eXtensible Markup Language (XML), the Simple Object Access Protocol (SOAP) and the Web Services Description Language (WSDL).

UDDI v.3 specifies a data model with four levels: business entities which provide services, for which bindings are described in terms of tModels. Note that there is a complete containment for the first three levels (business, service, binding) but not for the fourth - tModel - which is linked in by reference. This data model can be managed through an API covering methods for inquiry, publication, security, custody, subscription and value sets.

6.2 X.500 and LDAP

The X.500 directory service is a global directory service. Its components cooperate to manage information about objects such as countries, organizations, people, machines, and so on in a worldwide scope. It provides the capability to look up information by name (a white-pages service) and to browse and search for information (a yellow-pages service).

The information is held in a directory information base (DIB). Entries in the DIB are arranged in a tree structure called the directory information tree (DIT). Each entry is a named object and consists of a set of attributes.

The Lightweight Directory Access Protocol (LDAP) is a network protocol for accessing directories, based on X.500 which uses an internet transport protocol.

6.3 COrba Services: COS Naming and COS Trader

COS Naming and COS Trade are two services standardized by the Object Management Group OMG.

The COS Naming service provides a white-pages service through which most clients of an ORB-based system locate objects that they intend to use (make requests of). Given an initial naming context, clients navigate naming contexts retrieving lists of the names bound to that context.

The COS Trader is a yellow-page service that facilitates the offering and the discovery of instances of services of particular types. Advertising a capability or offering a service is called export. Matching against needs or discovering services is called import. Export and import facilitate dynamic discovery of, and late binding to, services.

Importers use a service type and a constraint to select the set of service offers in which they have an interest. The constraint is a well formed expression conforming to a constraint language. The operators of the language are comparison, boolean connective, set inclusion, substring, arithmetic operators, property existence. Then the result set is ordered using preferences before being returned to the importer. The ordering preference can be specified as either the maximization of minimization of a numeric expression, as the services that fulfill a given boolean constraint, as a random pick of the matching services or by default just as they are ordered by the trader.

When querying the trader results can be returned in chunks of a given size, specified by the importer.

6.4 FIPA Directories: AMS and DF

The FIPA management specification defines two kind of directories: the Agent Management System (white pages) and the Directory Facilitator (yellow pages). Both directories can be queried using a projection mechanism where a template of the objects of interest can be specified.

AMS entries define agent identification information, the state of the agent (active, suspended, etc.) and the ownership of the agent.

As specific functionality the AMS can be specified as transport resolver for given agent identifier entries. Upon submitting a search operation with an agent identifier without transport information the AMS will return another version of the agent identifier that will include transport details. Finally the AMS also is able to provides metadata regarding the current platform like name, transport addresses and platform-level services.

DF entries are more expressive: each entry defines one or more services provided by a given agent. Each service is describe trough the accepted content languages, ontologies and protocols. DF entries can be published on a leasing basis such that once a service provider fails to renew it's lease the entry will be removed from the DF. DFs can be federated and searched recursively. The search is controlled by simple parameters like search depth and maximum number of results. The results of the search are returned all at the time, in the internal order decided by the queried DF.

6.5 JINI Registrars

The JINI infrastructure provides object matching capabilities trough distribute directories named registrars. Service providers advertise their capabilities by leasing the publication of an advertisement entry to a previously discovered registrar. When the service provider fails to renew the lease it's advertisement is remove from the registry. Service consumers search the registrar by specifying entry templates.

Match operations use entry objects of a given type, whose fields can either have values (references to objects) or wildcards (null references). When considering a template T as a potential match against an entry E, fields with values in T must be matched exactly by the value in the same field of E. Wildcards in T match any value in the same field of E.

The results of the search are returned all at the time, in the internal order of the registrar.

7 Conclusions

In this paper we have looked at aspects specific to service integration in large open environments like the interplay between massive discovery and composition using partially matching services. We have described a service directory system and mechanisms specifically designed to meet the requirements of the assumed environment: numerically encoding of service descriptions, the use of indexing techniques that allow the efficient retrieval of matching services including the partially matching ones and finally a specific scheme for enabling service integration sessions. Sessions provide a constant view of the directory during the integration episode, the retrieval at each integration step of only unknown service descriptions and consistent concurrent accesses to the directory between different clients. We presented some experimental results on randomly generated problems that show mainly that the implemented system scales well and is able to deal with large numbers of services and also that partial matches bring significant gains in the range of problems that can be solved. Finally we have reviewed a number of existing directory systems that fulfills some but not all of the specific requirements for service integration. In particular by using sessions our system is able to provide unique features specific to service integration like consistent views during integration episodes and the retrieval at each step of only unknown services descriptions.

References

[1] D.-S. C. A. Ankolekar, M. Burstein, J. R. Hobbs, O. Lassila, D. Martin, D. McDermott, S. A. McIlraith, S. Narayanan, M. Paolucci, T. Payne, and K. Sycara. DAML-S: Web service description for the Semantic Web. *Lecture Notes in Computer Science*, 2342, 2002.

[2] B. Benatallah, Q. Z. Sheng, and M. Dumas. The self-serv environment for web services composition. *IEEE Internet Computing*, 7(1):40–48, 2003.

[3] A. L. Blum and M. L. Furst. Fast planning through planning graph analysis. *Artificial Intelligence*, 90(1–2):281–300, 1997.

[4] BPEL4WS. Business process execution language for web services version 1.1, http://www.ibm.com/developerworks/library/ws-bpel/.

[5] F. Casati, S. Ilnicki, L. Jin, V. Krishnamoorthy, and M.-C. Shan. Adaptive and dynamic service composition in eflow. Technical Report HPL-2000-39, Hewlett Packard Laboratories, Apr. 06 2000.
[6] I. Constantinescu and B. Faltings. Efficient matchmaking and directory services. In *The 2003 IEEE/WIC International Conference on Web Intelligence*, 2003.
[7] DAML-S. DAML Services, http://www.daml.org/services.
[8] FIPA. Foundation for Intelligent Physical Agents Web Site, http://www.fipa.org/.
[9] F. Giunchiglia and P. Traverso. Planning as model checking. In *European Conference on Planning*, pages 1–20, 1999.
[10] J. M. Hellerstein, J. F. Naughton, and A. Pfeffer. Generalized search trees for database systems. In U. Dayal, P. M. D. Gray, and S. Nishio, editors, *Proc. 21st Int. Conf. Very Large Data Bases, VLDB*, pages 562–573. Morgan Kaufmann, 11–15 1995.
[11] C. A. Knoblock, S. Minton, J. L. Ambite, N. Ashish, I. Muslea, A. Philpot, and S. Tejada. The Ariadne Approach to Web-Based Information Integration. *International Journal of Cooperative Information Systems*, 10(1-2):145–169, 2001.
[12] M. Kornacker, C. Mohan, and J. M. Hellerstein. Concurrency and recovery in generalized search trees. In J. M. Peckman, editor, *Proceedings, ACM SIGMOD International Conference on Management of Data: SIGMOD 1997: May 13–15, 1997, Tucson, Arizona, USA*, volume 26(2) of *SIGMOD Record (ACM Special Interest Group on Management of Data)*, pages 62–72, New York, NY 10036, USA, 1997. ACM Press.
[13] S. McIlraith, T. Son, and H. Zeng. Mobilizing the semantic web with daml-enabled web services. In *Proc. Second International Workshop on the Semantic Web (SemWeb-2001)*, Hongkong, China, May 2001.
[14] S. A. McIlraith and T. C. Son. Adapting golog for composition of semantic web services. In D. Fensel, F. Giunchiglia, D. McGuinness, and M.-A. Williams, editors, *Proceedings of the 8th International Conference on Principles and Knowledge Representation and Reasoning (KR-02)*, pages 482–496, San Francisco, CA, Apr. 22–25 2002. Morgan Kaufmann Publishers.
[15] M. Paolucci, T. Kawamura, T. R. Payne, and K. Sycara. Semantic matching of web services capabilities. In *Proceedings of the 1st International Semantic Web Conference (ISWC)*, 2002.
[16] B. Srivastav. Automatic web services composition using planning. In *International Conference on Knowledge Based Computer Systems (KBCS-2002)*, 2002.
[17] K. Sycara, J. Lu, M. Klusch, and S. Widoff. Matchmaking among heterogeneous agents on the internet. In *Proceedings of the 1999 AAAI Spring Symposium on Intelligent Agents in Cyberspace*, Stanford University, USA, March 1999.
[18] S. Thakkar, C. A. Knoblock, J. L. Ambite, and C. Shahabi. Dynamically composing web services from on-line sources. In *Proceeding of the AAAI-2002 Workshop on Intelligent Service Integration*, pages 1–7, Edmonton, Alberta, Canada, July 2002.
[19] UDDI. Universal Description, Discovery and Integration Web Site, http://www.uddi.org/.
[20] W3C. OWL web ontology language 1.0 reference, http://www.w3.org/tr/owl-ref/.
[21] W3C. Web services description language (wsdl) version 1.2, http://www.w3.org/tr/wsdl12.
[22] W3C. XML Schema, http://www.w3.org/xml/schema.
[23] Wu, Dan and Parsia, Bijan and Sirin, Evren and Hendler, James and Nau, Dana. Automating DAML-S Web Services Composition Using SHOP2. In *Proceedings of 2nd International Semantic Web Conference (ISWC2003)*, Sanibel Island, Florida, October 2003.
[24] J. Yang and M. P. Papazoglou. Web component: A substrate for Web service reuse and composition. *Lecture Notes in Computer Science*, 2348, 2002.