Searching for Services on the Semantic Web Using Process Ontologies

Mark Klein
Center for Coordination Science
Massachusetts Institute of Technology
m_klein@mit.edu

Abraham Bernstein
Stern School of Business
New York University
bernstein@stern.nyu.edu

Abstract. The ability to rapidly locate useful on-line services (e.g. software applications, software components, process models, or service organizations), as opposed to simply useful documents, is becoming increasingly critical in many domains. As the sheer number of such services increases it will become increasingly more important to provide tools that allow people (and software) to quickly find the services they need, while minimizing the burden for those who wish to list their services with these search engines. This can be viewed as a critical enabler of the 'friction-free' markets of the 'new economy'. Current service retrieval technology is, however, seriously deficient in this regard. The *information retrieval* community has focused on the retrieval of documents, not services per se, and has as a result emphasized keyword-based approaches. Those approaches achieve fairly high recall but low precision. The *software agents* and *distributed computing communities* have developed simple 'frame-based' approaches for 'matchmaking' between tasks and on-line services increasing precision at the substantial cost of requiring all services to be modeled as frames and only supporting perfect matches. This paper proposes a novel, ontology-based approach that employs the characteristics of a process-taxonomy to increase recall without sacrificing precision and computational complexity of the service retrieval process.

1 The Challenge

Increasingly, the Semantic Web will be called upon to provide access not just to static *documents* that collect useful *information*, but also to *services* that provide useful *behavior*. Potential examples of such services abound:

- *Software applications* (e.g. for engineering, finance, meeting planning, or word processing) that can invoked remotely by people or software
- *Software components* that can be downloaded for use when creating new applications
- *Process models* that describe how to achieve some goal (e.g. eCommerce business models, material transformation processes, etc)
- *Individuals* or *organizations* who can perform particular functions, e.g. as currently brokered using such web sites as guru.com, elance.com and freeagent.com.

As the sheer number of such services increase it will become increasingly important to provide tools that allow people (and software) to quickly find the services they need, while minimizing the burden for those who wish to list their services with these search engines [1]. This paper describes a set of ideas, based on the sophisticated use of process ontologies, for creating improved service retrieval technologies.

2 Contributions and Limitations of Current Technology

Current service retrieval approaches have serious limitations with respect to meeting the challenges described above. They either perform relatively poorly or make unrealistic demands of those who wish to index or retrieve services. We review these approaches below.

Service retrieval technology has emerged from several communities. The information retrieval community has focused on the retrieval of documents, not services per se, and has as a result emphasized keyword-based approaches. The software agents and distributed computing communities have developed simple ‘frame-based’ approaches for ‘matchmaking’ between tasks and on-line services. The software engineering community has developed by far the richest set of techniques for service retrieval [2]. We can get a good idea of the relative merits of these approaches by placing them in a *precision/recall* space (Figure 1):

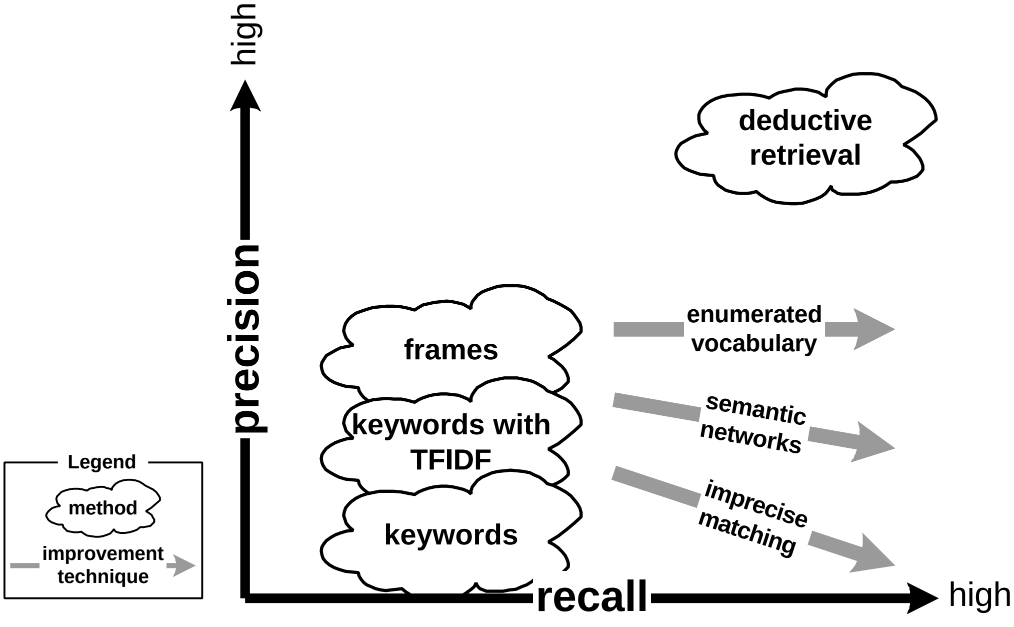

Figure 1: The state of the art in service retrieval.

Recall is the extent to which a search engine retrieves *all* of the items that one is interested in (i.e. avoiding false negatives) while *precision* is the extent to which the tool retrieves *only* the items that one is interested in (i.e. avoiding false positives).

Most search engines look for items (e.g. web pages) that contain the keywords in the query. More sophisticated variants (based on the technique known as TFIDF) look for items in which the searched-for keywords are more common than usual, thereby increasing precision [3]. Typically, no manual effort is needed to list items with such search engines, and queries can be specified without needing to know a specialized query language. Keyword-based approaches are, however, notoriously prone to both low precision and imperfect recall. Many completely irrelevant items may include the keywords in the query, leading to low precision. It is also possible that the query keywords are semantically equivalent but syntactically different from the words in the searched items, leading to reduced recall. Imagine , for example, that we are searching for a service that can offer a loan to cover a $100,000 house addition. Entering the keywords "loan house 100000" into google.com (a keyword-based search service), for example, returns 2,390 documents. While the first two results are promising (a loan calculator and a mortgage calculator somewhat connected to a loan granting organization), the third hit points to a report of a campaign against arms trade, and the fourth shows a junior high-school math-project on how to calculate mortgages. If we enter the same query in altavista.com (which uses TFIDF) we get 24,168,519 'hits'. While the first few hits talk about loans, they do not all provide a loan service. Most of them point to classes that discuss loan calculation techniques and provide some sort of a mortgage calculator.

It is of course somewhat misleading to use web-search engines to assess the likely performance of a keyword-based service retrieval engine. The web contains much more then services. Many documents (like the loan calculation classes mentioned above) have nothing to do with the provision of services. Nevertheless we can take the poor precision of those queries as an indicator of what would happen in a system that relies solely on these techniques for retrieving services. A mortgage calculator is a useful instrument in itself. It does not, however, provide the service of creditworthiness analysis or loan provision. We can, therefore, assume that systems using these techniques would still show low precision.

Several techniques have been developed to address these problems. One is to require that items and queries be described using the same, pre-enumerated, vocabulary [4]. This increases the probability that the same terms will be used in the query and desired items, thereby increasing recall. Another approach is to use semantic nets (e.g. WordNet [5]) that capture the semantic relationships between words (e.g. synonym, antonym, hypernym and hyponym) to increase the breadth of the query and thereby increase recall [6] but this potentially can reduce precision and can result in suboptimal recall because these networks focus on purely linguistic relationships. Search engines like google.com [7] prioritize retrieved documents according to whether they are linked to documents that also contain the searched-for keywords, as a way of increasing precision. Finally, most text-based search engines allow for imprecise matching (e.g. retrieving items that contain some but not all of the query keywords), potentially increasing recall but again at the cost of reduced precision.

We can see then that keyword-based approaches can achieve fairly high recall but at the cost of low precision. The key underlying problem is that keywords are a poor way to capture the semantics of a query or item. If this semantics could be captured more accurately then precision would increase. *Frame*-based approaches [8] [9] [10] [11] [12] have emerged as a way of doing this. A frame consists of attribute value pairs describing the properties of an item. Figure 2 for example shows a frame for an integer averaging service:

Both items and queries are described using frames: matches represent items whose (textual) property values match those in the query. All the commercial service search technologies we are aware of (e.g. Jini, eSpeak, Salutation, UDDI [13]) use the frame-based approach, typically with an at least partially pre-enumerated vocabulary of service types and properties. The more sophisticated search tools emerging from the research community (e.g. LARKS [14]) also make limited use of semantic nets, e.g. returning a match if the input type of a service is equal to *or* a generalization of the input type specified in the query. Frame-based approaches thus do increase precision at the (fairly modest) cost of requiring that all services be modeled as frames.

Description	a service to find the average of a list of integers
Input	integers
Output	real
Execution Time	number of inputs * 0.1 msec

Figure 2: A frame-based description of an integer sorting service.

The frame-based approach is taken one step further in the *deductive retrieval* approach [15] [16] [17] wherein service properties (e.g. inputs, outputs, function, and performance) are expressed *formally* using logic (Figure 3):

Name:	set-insert
Syntax:	set-insert(Elem, Old, New)
Input-types:	(Elem:Any), (Old:SET)
Output-types:	(New: SET)
Semantics:	
Precond:	$\neg member(Elem, Old)$
	$member(Elem, New)$
Postcond:	$\wedge \forall x(member(x, Old) \rightarrow member(x, New)$
	$\wedge \forall y(member(y, New) \rightarrow (member(y, old) \vee y = Elem)$

Figure 3: A service description for deductive retrieval [15]

Retrieval then consists of finding the items that can be *proved* to achieve the functionality described in the query. If we assume a non-redundant pre-enumerated vocabulary of logical predicates and a complete formalization of all relevant service and query properties, then deductive retrieval can in theory achieve both perfect precision and perfect recall. This approach, however, faces two very serious practical difficulties. First of all, it can be prohibitively difficult to model the semantics of non-trivial queries and services using formal logic. Even the simple set-insert function shown above in Figure 3is non-trivial to formalize correctly: imagine trying to formally model the behavior of Microsoft Word or an accounting package! The second difficulty is that the proof process implicit in this kind of search can have a high computational complexity, making it extremely slow [15]. Our belief is that these limitations, especially the first one, make deductive retrieval unrealistic as a scalable general purpose service search approach.

Other approaches do exist, but they apply only to specialized applications. One is execution-based retrieval, wherein software components are selected by comparing their actual I/O behavior with the desired I/O behavior. This is approach is suitable only for contexts where observing a few selected samples of I/O behavior are sufficient to prune the service set [18] [19] [20].

3 Our Approach: Exploiting Process Ontologies

Our challenge, as we have seen, can be framed as being able to capture enough service and query semantics to substantively increase precision without reducing recall or making it unrealistically difficult for people to express these semantics. *Our central claim is that these goals can be achieved through the sophisticated use of process ontologies*. We begin by capturing the function(s) of a service as a process model. The service model is then indexed (to facilitate its subsequent retrieval) by placing it and all its components (subtasks and so on) into the appropriate sections of the ontology. Queries are also expressed as (partial) process models. The matching algorithm then finds all the services whose process models match that of the query, using the semantic relationships encoded in the process ontology. Our approach can thus be viewed as having the following functional architecture:

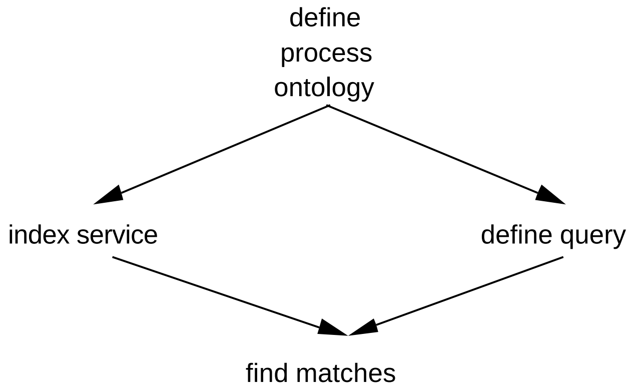

Figure 4: Functional architecture of our proposed service retrieval technology.

We will consider each element of the functional architecture in the sections below.

3.1 *Define Process Ontology*

Our approach differs from previous efforts in that it is based on highly expressive process models arranged into a fully-typed process ontology. The key concepts underlying this ontology are an extension of those developed by the MIT Process Handbook project. The Handbook is a process knowledge repository which has been under development at the Center for Coordination Science (CCS) for the past eight years [21] [22]. The Handbook is under active use and development by a highly distributed group of more than 40 scientists, teachers, students and sponsors for such diverse purposes as adding new process descriptions, teaching classes, and business process re-design. We believe the current Handbook process ontology represents an excellent starting point for indexing on-line services because it is focused on business processes which is what a high proportion of on-line services are likely to address.

The Handbook takes advantage of several simple but powerful concepts to capture and organize process knowledge: *attributes, ports, decomposition, dependencies, exceptions* and *specialization.*

> *Process Attributes:* Like most process modeling techniques, the Handbook allows processes to be annotated with attributes that capture such information as a textual

description, typical performance values (e.g. how long a process takes to execute), as well as pre-, post- and during- conditions.

Decomposition: Also like most process modeling techniques, the Handbook uses the notion of *decomposition*: a process is modeled as a collection of activities that can in turn be broken down ("decomposed") into subactivities.

Ports: Ports describe the I/O-behavior of an activity. They describe the types of resources the activity uses and produces, and are as a result important for assessing the match between a service specification and a query.

Dependencies: Another key concept we use is that coordination can be viewed as the management of *dependencies* between activities [21]. Every dependency can include an associated *coordination mechanism*, which is simply the process that manages the *resource* flow and thereby coordinates the activities connected by the dependency. Task inputs and outputs are represented as *ports* on those tasks. A key advantage of representing processes using these concepts is that they allow us to highlight the 'core' activities of a process and abstract away details about how they coordinate with each other, allowing more compact service descriptions without sacrificing significant content.

Exceptions: Processes typically have characteristic ways they can fail and, in at least some cases, associated schemes for anticipating and avoiding or detecting and resolving them. This is captured in our approach by annotating processes with their characteristic 'exceptions', and mapping these exceptions to processes describing how these exceptions can be handled [23].

Specialization: The final key concept is that processes and all their key elements (ports, resources, attributes, and exceptions) appear in type taxonomies, with very generic classes at one extreme and increasingly *specialized* ones at the other, so process models are fully typed. The taxonomies place items with similar semantics (e.g. processes with similar purposes) close to each other, the way books with similar subjects appear close to each other in a library: Processes that vary along some identifiable dimension can be grouped into *bundles*; where processes can appear in more than one bundle. Bundles subsume the notion of 'faceted' classification [4], well-known in the software component retrieval community, because bundles, unlike facets, can include a whole ontology branch as opposed to simply a flat set of keywords, allowing varying abstraction levels and the use of synonyms.

As shown in Figure 5, an activity is thus defined by specifying its decomposition (i.e., the activities it contains), its interface (as defined by the ports it contains), the dependencies between its sub-activities, and the attributes defined for each of those entities (not shown). Each activity can be linked to the kinds of exceptions it can face, and these exceptions can be linked in turn to the processes if any used to handle (anticipate and avoid, or detect and resolve) them.

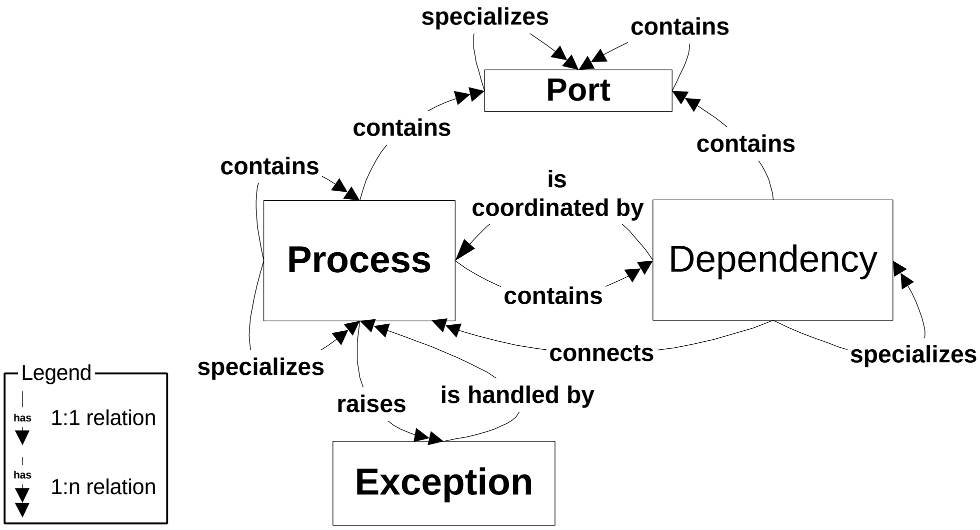

Figure 5: Partial meta-model for our process ontology (attributes not shown).

Every type of entity has its own specialization hierarchy into which it can be placed, making this a fully-typed process description approach. There is thus a specialization hierarchy for processes, resources, exceptions and so on. The “Sell loan” process, for example, is a specialization of the more general “Sell financial service” process. It, furthermore, specializes into more specific processes such as “Sell reserve credit,” “Sell Credit Card,” and “Sell mortgage” (Figure 6):

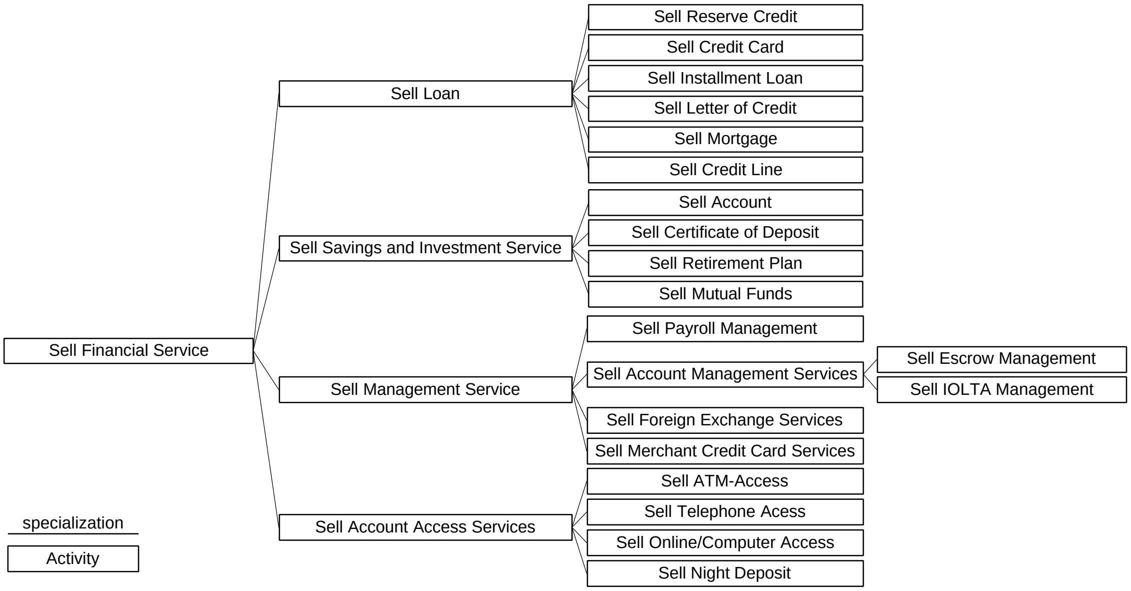

Figure 6: Specialization hierarchy for grant loan

Each of the elements of the “Sell loan” process is also a member of the overall specialization hierarchy. “Analyze credit-worthiness,” for example, is a specialization of the more general

"Perform financial analysis" and specializes to more specific processes such as "Analyze creditworthiness using scoring" and "Analyze creditworthiness using PE-ratio," etc.

This process representation has equal or greater formal expressiveness than other full-fledged process modeling languages (e.g. IDEF [24], PIF [25], PSL [26] or CIMOSA [27]) as well as greater expressiveness than the frame-based languages used in previous service retrieval efforts, by virtue of adding such important concepts as full typing, resource dependencies, ports, task decompositions, and exceptions.

The growing Handbook database currently includes over 5000 process descriptions ranging from specific (e.g. for a university purchasing department) to generic (e.g. for resource allocation and multi-criteria decision making). A subset of this database (containing a representative selection of process models but no exception-related information) is accessible over the Web at http://ccs.mit.edu/eph/

3.2 *Index Services*

Services are indexed into the process ontology so that they may be retrieved readily later on. Indexing a service in our approach comes down to placing the associated process model, and all of its components (attributes, ports, dependencies, subtasks and exceptions) in the appropriate place in the ontology. The fact that a substantial process ontology exists in the Process Handbook means that parties wishing to index a service can construct their service specification from already existing elements in the ontology, and then customizing them as necessary.

Imagine for example that we want to index a service that sells mortgages. The Handbook ontology already includes a general 'sell loan' process (Figure 7):

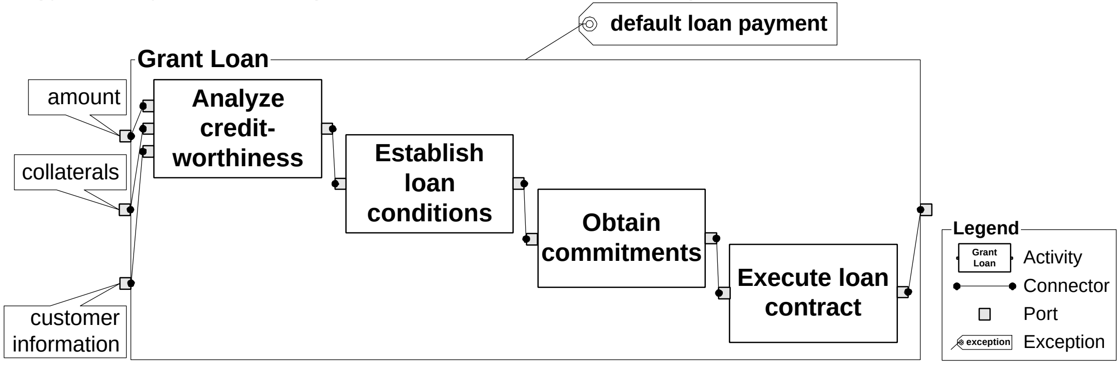

Figure 7: The loan selling service process model.

The 'sell loan' process model includes most of the requisite elements, including key sub-activities such as analyzing creditworthiness, establishing the loan's conditions, obtaining the commitments, and "executing" the loan contract. It also includes ports describing the process inputs (e.g. the size of the loan ("amount"), the collateral provided for the loan, information about the customer) and outputs (the loan itself).[1] The mortgage provider, therefore, need only create a specialization of the 'sell loan' process and make the few changes specific to mortgage loans, e.g. further specifying the types of customer information it wishes and type of creditworthiness analysis performed, elements that can also be defined by specializing existing elements in the process ontology. (Figure 8):

[1] To simplify this example we omitted dependencies and exceptions.

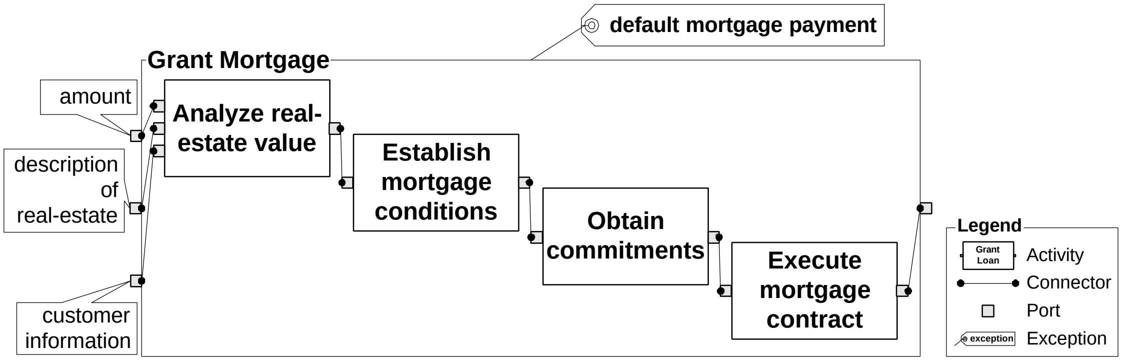

Figure 8: The mortgage selling service process model.

The Handbook project has developed sophisticated Windows-based tools for process indexing, based on this define-by-specialization principle, that have been refined over nearly eight years of daily use. Using these tools, most process models can indexed in a matter of minutes.

Note that when modeling a service, one needs in theory to account for all the possible uses the service may be put to. What to some users may be a side effect of a service may be a key attribute for others. To pick a homely example, a dog-walking service may be desired as a way to provide the dog exercise, or simply as a way to make sure the dog is cared for while the owner is out on an errand. This would suggest that a process model needs to be indexed under specializations representing all of its possible uses, which is difficult at best. As we shall see below, however, we can define query algorithms that, using query mutation operators, can potentially retrieve processes not explicitly indexed as serving a given purpose, thereby reducing the service-indexing burden.

We can, however, take the manual indexing approach described above one important step further. A key criterion for a successful service retrieval approach is minimizing the manual effort involved in listing new services with the search engine. Ideally services can be classified automatically. Automatic classification is predicated on having a similarity metric so one can place a process under the class it is most similar to. Previous efforts for automatic service classification have used similarity metrics based on either:

- Automatically calculated word frequency statistics for the natural language documentation of the service [28] [29] [30]
- Manually developed frame-based models of the service [9]

The first approach is more scalable because no manual effort is needed, but the latter approach results in higher retrieval precision because frame-based models, as we have seen, better capture service semantics. We can use a ontology-based approach to improve on these approaches by developing tools for [semi-] automated classification of process models. Process models, in the form of flowcharts for example, are typically created as part of standard programming practice, so in many if not most cases the services we want to index will already have process models defined for them. These models may even in some cases already be classified into some kind of process ontology. The greater expressiveness of process models as compared to standard frame-based service models (process models often include task decomposition information, for example, but frame-based models do not) means that we can define similarity metrics that are more accurate than those that have been used previously. These metrics can use word frequency statistics augmented with semantic networks to compute similarity of natural language elements, combined with taxonomic reasoning and graph-theoretic similarity measures to assess structural and type similarities. At the least, we believe that this technique should greatly speed manual service indexing by providing the human user with a short list of possible locations for a service. Ideally, it will allow a human user to be taken out of the loop entirely. If fully automated indexing turns out to be feasible, then we can imagine on-line services being indexed using Internet "worms" analogous to those currently used by keyword-based search tools.

3.3 *Define Queries*

It is of course possible that we can do without query definition entirely once services have been indexed into a process ontology. In theory one can simply browse the ontology to find the services that one is interested in, as in [31]. Our own experience with the Process Handbook suggests however that browsing can be slow and difficult for all except the most experienced human users, because of the size of the process ontology. This problem is likely to be exacerbated when, as with online services, the space of services is large and dynamic.

To address this challenge we have begun to define a query language called PQL (the Process Query Language) designed specifically for retrieving full-fledged process models from a process ontology [32]. Process models can be straightforwardly viewed as entity-relationship diagrams made up of entities like tasks connected by relationships like 'has-subtask'. PQL queries are built up as combinations of two types of clauses that check for given entities and relationships:

```
Entity <entity> isa <entity type>  :test <predicate>
Relationship <source entity> <relationship type>  <target entity>  :test <predicate>
```

The first clause type matches any entity of a given type. The second primitive matches any relationship of a given type between two entities. Any bracketed item <> can be replaced by a variable (with the format ?<string>) that is bound to the matching entity and passed to subsequent query clauses. The predicates do further pruning of what constitutes a match.

Let us consider some simple examples. Imagine we are searching for a loan service that accepts real estate as collateral (which includes mortgages), as follows:

```
Entity ?proc isa sell-loan                               // finds a specialization of “Sell loan”

Relationship ?proc has-port ?port1                       // looks for a port that accepts real-
Entity ?port1 isa input-port                             // estate as a collateral
Relationship ?port1 propagates-resource ?res1
Entity ?res1 isa real-estate
Entity ?port1 has-attribute ?attr1
Entity ?attr1 isa name
Relationship ?attr1 has-value ?val :test (= ?val “collateral”)

Relationship ?proc has-port ?port2                       // looks for a port that “produces”
Entity ?port2 isa output-port                            // money
Relationship ?port2 propagates-resource ?res2
Entity ?res2 isa money
```

Query 1: A query for mortgage services.

This type of query is imaginably within the capabilities of frame-based query languages, especially if enhanced with a semantic network such as WordNet [5], since it references only the type and I/O for a service. Existing linguistically-oriented semantic networks do not, however, include a service-oriented process taxonomy like that incorporated in the Handbook, so we can expect that recall will be less than optimal using this approach.

We can go substantially further, however, using an ontology-based approach. Query 2 below, for example, retrieves a financing solution that has at least one internet-based step:

```
Entity ?proc isa sell-loan                               // finds a specialization of “Sell loan”

Relationship ?proc has-subtask ?sub                      // has sub-process that is internet
Entity ?sub isa internet-process                         // based

Relationship ?proc has-port ?port1                       // looks for a port that accepts real-
Entity ?port1 isa input-port                             // estate as a collateral
Relationship ?port1 propagates-resource ?res1
Entity ?res1 isa real-estate
Entity ?port1 has-attribute ?attr1
Entity ?attr1 isa name
Relationship ?attr1 has-value ?val :test(= ?val “collateral”)

Relationship ?proc has-port ?port2                       // looks for a port that “produces”
Entity ?port2 isa output-port                            // money
Relationship ?port2 propagates-resource ?res2
Entity ?res2 isa money
```

Query 2: A query for internet-based financing services.

This useful capability – searching for services based on *how* they achieve their purpose – can *not* be achieved using frame-based languages since they do not capture process decompositions.

Another novel and powerful feature of the ontology-based approach is the ability to search based on how the service handles *exceptions*. Query 2 could be refined, for example, by adding a clause that searches for loan processes that provide insurance for handling payment defaults:

```
Entity ?proc isa sell-loan
…
Relationship ?proc has-exception ?exc
Entity ?exc isa payment-default
Relationship ?exc is-handled-by ?proc2
Entity ?proc2 isa insurance-process
```

Query 3: A query refinement that searches for payment default insurance.

Searching based on a services' exception handling processes is also not well-supported by frame-based techniques. We can see, in short, that an ontology-based approach offers substantively greater query expressiveness than keyword or frame-based approaches, by virtue of having a substantive process taxonomy, and by being able to represent process decompositions and exception handling.

PQL, like any query language, is fairly verbose and requires that users be familiar with its syntax. More intuitive interfaces suitable for human users are possible. One approach exploits the process ontology. It is straightforward to translate any process model into a PQL query that looks for a service with that function. Users can thus use the Handbook process modeling tools to express a query as a combination and refinement of existing elements in the process ontology, in much the same way new services are indexed. An advantage of specifying a query in this way is that the elements in the process ontology can give the user additional ideas concerning what to look for. Someone looking for a mortgage service, for example, might not initially think to check for whether or not that service provides mortgage insurance. If they define the query as a specialization of the generic 'sell loan' process, however, they may note that that process has a possible 'payment default' exception, and they mya as a result be inspired to search for mortgage services that handle that exception in a way the user prefers.

Other possibilities exist. One can reduce the query definition burden by allowing users to enter queries using a restricted subset of English (an approach that has been applied successfully to traditional database access). Substantial progress has also been made in defining graphical metaphors for query definition (e.g. see [33]). These approaches can of course be combined, so that for example a simple natural language query returns some candidates classes that are selected from and refined to define the final, more complete, query.

3.4 *Find Matches*

The algorithm for retrieving matches given a PQL query is straightforward. The clauses in the PQL query are tried in order, each clause executed in the variable binding environment accumulated from the previous clauses. The sets of bindings that survive to the end represent the matching services. There is one key problem, however, that has to be accounted for; what we can call *modeling differences*. It is likely that in at least some cases a service may be modeled in a way that is semantically equivalent to but nevertheless does not syntactically match a given PQL query. The service model may, for example, include a given subtask several levels down the process decomposition, while in the query that subtask may be just one level down. The service model may express using several resource flows what is captured in the query as a single more abstract resource flow. The service model may simply be missing some type or resource flow information tested for in the query. This problem is exacerbated by the possibility of multiple service ontologies. As has been pointed out in [34], while we can expect the increasing prevalence of on-line ontologies to structure all kinds of knowledge including service descriptions, there will almost certainly be many partially-mapped ontologies as opposed to a single universally adopted one. This will likely increase the potential for modeling differences, e.g. if queries and services are defined using different ontologies. In order to avoid false negatives we must therefore provide a retrieval scheme that is tolerant of such modeling differences.

We can explore for this purpose the use of semantics-preserving query mutation operators. Imagine we have a library of operators that can syntactically mutate a given PQL query in a way that (largely) preserves its semantics. Some examples of such operators include:

(a) allow a type specification to be more general
(b) allow a subtask to be any number of levels down the task decomposition hierarchy
(c) allow ‘siblings’ or ‘cousins’ of a task to constitute a match
(d) relax the constraints on a parameter value
(e) remove a subtask

One can use mutation operators to broaden a search to allow for the discovery of novel alternatives. One could for example apply mutation operator (a) to Query 4 above so that it searches for “Sell” processes, not just “Sell loan” processes. This would result in additional hits like “Sell real-estate,” a service which would sell the property to raise money, or “Sell real-estate lease,” which would lease the property to generate funds. Assuming a homeowner would want to raise money for an extension, the first of those two additional options would not be desirable, as the property would be sold off. The second option, though, is often used. The key point to be made here is that the mutation operation broadened the search in a way that was *semantically motivated*, thereby avoiding the precipitous decline in precision typical of other imprecise matching techniques.

A second important potential use of mutation operators is to address the incomplete indexing issue described in section 3.2. It will often be the case that a user may be searching for all processes that serve a purpose not represented by any single element in the process ontology. For example, one may be searching for all the ways to raise money, but the different options may not all share a single generalization. One can address this issue by using mutation operators to define a query that searches for processes with *similar substeps*, regardless of what part of the process ontology they are indexed in. So we can, for example, search for all processes that offer money given some kind of collateral. This would return such options as services for getting a mortgage, raising money on the stock market, and so on.

4 Contributions of this Work

This paper has described a set of ideas that exploit the use of process model representations of service semantics, plus process ontologies, to improve service retrieval. These ideas offer the potential for the following important benefits:

Increased Precision: Our approach differs from previous efforts in that it models service semantics more fully than keyword or frame-based approaches, without imposing the unrealistic modeling burden implied by deductive retrieval approaches. This translates into greater retrieval precision.

Increased Recall: Modeling differences between queries and service descriptions can reduce recall, but this can be addressed in a novel way through the use of semantics-preserving query mutation operators.

While query definition and service indexing using process models is potentially more burdensome than simply entering a few keywords, we believe that existing "define-by-specialization" process modeling techniques developed in the Handbook project, coupled with our proposed advances in search algorithms and automated process model classification, should result in an acceptable increase in the query definition and service indexing burden when traded off against the increase in retrieval precision and recall.

To date we have developed an initial version of a PQL interpreter as well as a small set of semantics-preserving query mutation operators [31], which has demonstrated the viability of the query-by-process-model concept. While we have not yet evaluated PQL's performance in detail yet, it is clear that its primitive query elements have low computational complexity. Query performance can in addition be increased by using such well-known techniques as query clause re-ordering. Our future efforts will involve comparing the precision and recall of our approach with other search engines, refining the query definition and mutation schemes, and implementing and evaluating automated process classification techniques.

5 References

1. Bakos, J.Y., *Reducing Buyer Search Costs: Implications for Electronic Marketplaces.* Management Science, 1997. **43**.
2. Mili, H., F. Mili, and A. Mili, *Reusing software: issues and research directions.* IEEE Transactions on Software Engineering, 1995. **21**(6): p. 528-62.
3. Salton, G. and M.J. McGill, *Introduction to modern information retrieval*. McGraw-Hill computer science series. 1983, New York: McGraw-Hill. xv, 448.
4. Prieto-Diaz, R., *Implementing faceted classification for software reuse.* 12th International Conference on Software Engineering, 1990. **9**: p. 300-4.
5. Magnini, B., *Use of a lexical knowledge base for information access systems.* International Journal of Theoretical & Applied Issues in Specialized Communication, 1999. **5**(2): p. 203-228.
6. Brin, S. and L. Page. *The anatomy of a large-scale hypertextual Web search engine*. in *Computer Networks & ISDN System*. 1998. Netherlands: Elsevier.
7. Henninger, S., *Information access tools for software reuse.* Journal of Systems & Software, 1995. **30**(3): p. 231-47.
8. Fernandez-Chamizo, C., et al., *Case-based retrieval of software components.* Expert Systems with Applications, 1995. **9**(3): p. 397-421.
9. Fugini, M.G. and S. Faustle. *Retrieval of reusable components in a development information system*. in *Second International Workshop on Software Reusability*. 1993: IEEE Press.

10. Devanbu, P., et al., *LaSSIE: a knowledge-based software information system.* Communications of the ACM, 1991. **34**(5): p. 34-49.
11. ESPEAK, *Hewlett Packard's Service Framework Specification*. 2000, HP Inc.
12. Richard, G.G., *Service advertisement and discovery: enabling universal device cooperation.* IEEE Internet Computing, 2000. **4**(5): p. 18-26.
13. Sycara, K., et al. *Matchmaking Among Heterogeneous Agents on the Internet*. in *AAAI Symposium on Intelligent Agents in Cyberspace*. 1999: AAAI Press.
14. Meggendorfer, S. and P. Manhart. *A Knowledge And Deduction Based Software Retrieval Tool*. in *6th Annual Knowledge-Based Software Engineering Conference*. 1991: IEEE Press.
15. Chen, P., R. Hennicker, and M. Jarke. *On the retrieval of reusable software components.* in *Proceedings Advances in Software Reuse. Selected Papers from the Second International Workshop on Software Reusability*. 1993.
16. Kuokka, D.R. and L.T. Harada, *Issues and extensions for information matchmaking protocols.* International Journal of Cooperative Information Systems, 1996. **5**: p. 2-3.
17. Podgurski, A. and L. Pierce, *Retrieving reusable software by sampling behavior.* ACM Transactions on Software Engineering & Methodology, 1993. **2**(3): p. 286-303.
18. Hall, R.j. *Generalized behavior-based retrieval (from a software reuse library)*. in *15th International Conference on Software Engineering*. 1993.
19. Park, Y., *Software retrieval by samples using concept analysis.* Journal of Systems & Software, 2000. **54**(3): p. 179-83.
20. Malone, T.W. and K.G. Crowston, *The interdisciplinary study of Coordination. ACM Computing Surveys*, 1994. **26**(1): p. 87-119.
21. Malone, T.W., et al., *Tools for inventing organizations: Toward a handbook of organizational processes.* Management Science, 1999. **45**(3): p. 425-443.
22. Klein, M. and C. Dellarocas, *A Knowledge-Based Approach to Handling Exceptions in Workflow Systems.* Journal of Computer-Supported Collaborative Work. Special Issue on Adaptive Workflow Systems., 2000. **9**(3/4).
23. NIST, *Integrated Definition for Function Modeling (IDEF0)*. 1993, National Institute of Standards and Technology.
24. Lee, J., et al. *The PIF Process Interchange Format and Framework Version 1.1*. in *European Conference on AI (ECAI) Workshop on Ontological Engineering*. 1996. Budapest, Hungary.
25. Schlenoff, C., et al., *The essence of the process specification language.* Transactions of the Society for Computer Simulation, 1999. **16**(4): p. 204-16.
26. Kosanke, K., *CIMOSA: Open System Architecture for CIM*. 1993: Springer Verlag.
27. Frakes, W.b. and B.a. Nejmeh, *Software reuse through information retrieval.* Proceedings of the Twentieth Hawaii International Conference on System Sciences, 1987. **2**: p. 6-9.
28. Maarek, Y.s., D.M. Berry, and G.e. Kaiser, *An information retrieval approach for automatically constructing software libraries.* IEEE Transactions on Software Engineering, 1991. **17**(8): p. 800-13.
29. Girardi, M.R. and B. Ibrahim, *Using English to retrieve software.* Journal of Systems & Software, 1995. **30**(3): p. 249-70.

30. Latour, L. and E. Johnson. *Seer: a graphical retrieval system for reusable Ada software modules*. in *Third International IEEE Conference on Ada Applications and Environments*. 1988: IEEE Comput. Soc. Press.

31. Klein, M. *An Exception Handling Approach to Enhancing Consistency, Completeness and Correctness in Collaborative Requirements Capture*. 1996.

32. Hendler, J., *Agents on the Semantic Web*. 2001.