Towards High-Precision Service Retrieval

Mark Klein
Massachusetts Institute of Technology
Room NE20-336
Cambridge, MA 02139, USA
m_klein@mit.edu

Abraham Bernstein
University of Zurich
Department of Information Technology
Winterthurerstr. 190
CH-8057 Zürich, Switzerland
Bernstein@ifi.unizh.ch

Abstract: Increasingly, on-line repositories such as the World Wide Web are being called upon to provide access not just to documents that collect useful *information*, but also to *services* (such as software, process models or even organizations) that describe or provide useful *behavior*. 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. Current service retrieval approaches have, however, serious limitations with respect to meeting these challenges. They either perform relatively poorly or make unrealistic demands of those who wish to index or retrieve services. This paper reviews these efforts and presents a novel approach, based on the use of process models, that offers qualitatively higher retrieval precision, acceptable computational complexity, and a relatively low service modeling burden.

Keywords: process models, service/resource discovery

1. The Challenge: High Precision Service Retrieval

Increasingly, on-line repositories such as those available via the World Wide Web are being called upon to provide access not just to documents that collect useful *information*, but also to *services* that describe or even provide useful *behavior*. Examples of such services include software applications and components (e.g. www.mibsoftware.com and www.compoze.com), best practice process models (e.g. process.mit.edu/eph/, www.brint.com and www.bmpcoe.com), and even individuals or organizations who can perform particular functions (e.g. guru.com and elance.com).

As the sheer number of such services increases 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]. Current service retrieval approaches have, however, serious limitations with respect to meeting these challenges. They either perform relatively poorly or make unrealistic demands of those who wish to index or

retrieve services. This paper reviews these approaches and presents a novel service retrieval approached based on the sophisticated use of process models.

2. The State of the Art

Service retrieval technology has emerged from several communities. The information retrieval community has focused on the retrieval of natural language documents, not services per se, and has as a result emphasized keyword-based as well as, more recently, concept-based approaches. The software agents and distributed computing communities have developed simple 'table-based' approaches for 'matchmaking' between tasks and on-line services. The software engineering community has also developed a rich set of techniques for service retrieval, most notably deductive retrieval. We can get a good idea of the relative merits of these approaches by placing them in a *precision/recall* space:

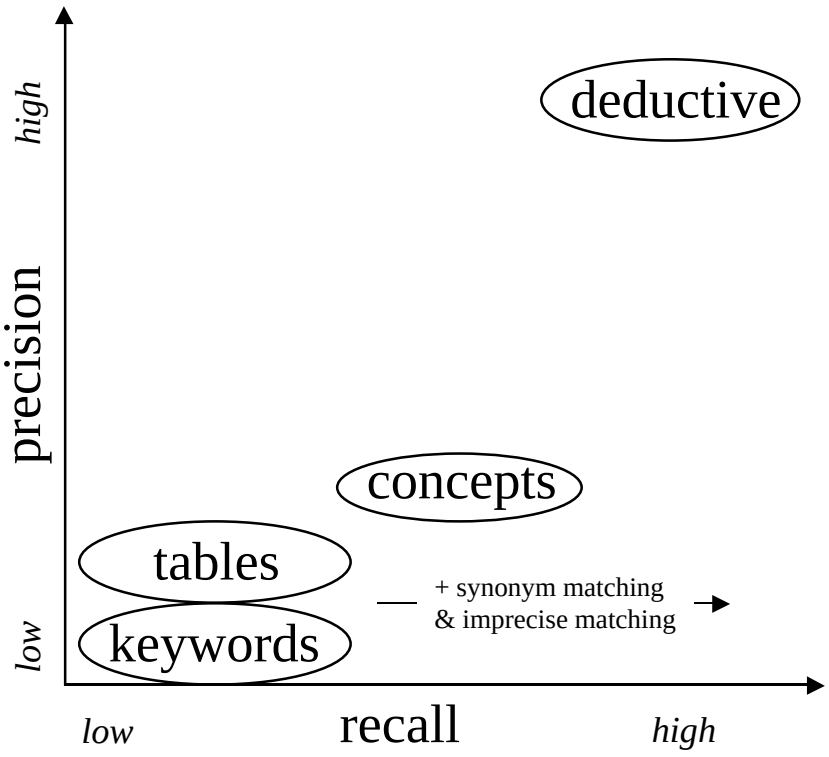

Figure 1. The state of the art in service retrieval.

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 that contain the *keywords* in the query. While services can be indexed with no manual effort, such approaches are 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. While techniques (e.g. synonym databases [2]) and imprecise matching are available to increase recall, these typically reduce precision even further. The key underlying problem is that keywords are a poor way to capture the semantics of a query or service. If this semantics could be captured more accurately then precision would increase.

Concept-based retrieval relies on defining an ontology of concepts that is used to classify documents, thereby enabling retrieval based on semantics rather than keywords [3] [4] [5] [6]. This approach potentially enables both increased precision and increased recall, but requires solutions for problems, such as automated ontology creation and document classification, that are extremely difficult and remain highly problematic.

Table-based approaches have emerged as a way of more fully capturing service semantics. A table-based service model consists of attribute value pairs that capture service properties, typically including its name, description, inputs and outputs, as well as some performance-related attributes such cost, execution time, and so on. The following, for example, is a table-based model for an integer averaging service:

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

Both items and queries are described as tables: matches represent items whose property values match those in the query. All the commercial service search technologies we are aware of (e.g. Jini, eSpeak [7], and UDDI [8]) use the table-based approach. Table-based approaches do increase precision, but only to a modest extent, because of the impoverished range of information they capture. These models typically include a detailed description of how to *invoke* the service (i.e., parameter types, return types, calling protocols, etc.), but don't describe what the service actually *does*, aside from an optional full-text description. The invocation-related information is of limited value for search purposes because services with different goals (e.g. services that compute averages and medians) can share identical call signatures.

A third important class of search technique is *deductive retrieval* [9] wherein service semantics are expressed formally using logic (Figure 2):

Name:	set-insert
Syntax:	set-insert(Elem, Old, New)
Input-types:	(Elem:Any), (Old:SET)
Output-types:	(New: SET)
Precond:	$\downarrow member(Elem, Old)$
	$member(Elem, New)$
Postcond:	$\hat{0}\ \ x(member(x, Old) \blacklozenge\ \ member(x, New)$
	$\hat{0}\ \ y(member(y, New) \blacklozenge\ \ (member(y, old) \int\ \ y = Elem)$

Figure 2. A logic-based service description [10]

Retrieval then consists of finding the items that can be *proved* to achieve the functionality described in the query. 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 2 is 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 [10]. Our belief is that these limitations, especially the first one, make deductive retrieval unrealistic as a scalable general purpose service search approach.

3. Our Approach: Exploiting Process–Based Service Models

Our challenge can thus 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 use of process models*. A process model captures behavior, such as what a service does, as a collection of interlinked sub-activities. The greater expressiveness of process models, as compared to keywords or tables, enables qualitatively increased retrieval precision, and we will argue that this can be achieved with a reasonable expenditure of service modeling effort. We describe the approach below. We consider first how to model service semantics using process models, describe the PQL query language, and then outline the algorithm used to find matches between process-based service models and these PQL queries.

Modeling Services as Process Models: The first step in our approach is to capture service behavior as process models. To understand why, we need to better understand the causes of imperfect precision (i.e. false positives). One cause is that a component of the service model is taken to have an unintended role. For example, a keyword-based query to find mortgage services that deal with "payment defaults" (a kind of exception) would also match descriptions like "the payment defaults to $100/month" (an attribute value). The other cause for false positives occurs when a service model is taken to include an unintended relationship between components. For example, we may be looking for a mortgage service where insurance is provided for payment defaults, but a keyword search would not distinguish this from a service that provides insurance for the home itself.

The trick to increasing retrieval precision, therefore, comes down to ensuring that the roles and relationships that are meaningful to the user are made explicit in both the query and the service model, so unintended meanings (and therefore false positives) can be avoided. We believe that process-modeling languages are well suited for this. Process modeling languages have been designed to capture the essence of different behaviors in a compact intuitive way, and have become ubiquitous for a very wide range of uses. The examples below offer concrete glimpses into how well suited process models are for capturing the semantics relevant for useful queries.

Broad consensus has emerged on how to model processes, using such primitives as tasks, resources, inputs, outputs, and exceptions. Our representation (formally described in [Berstein, 2002 #4272]) is a straightforward formalization of this consensus (Figure 3):

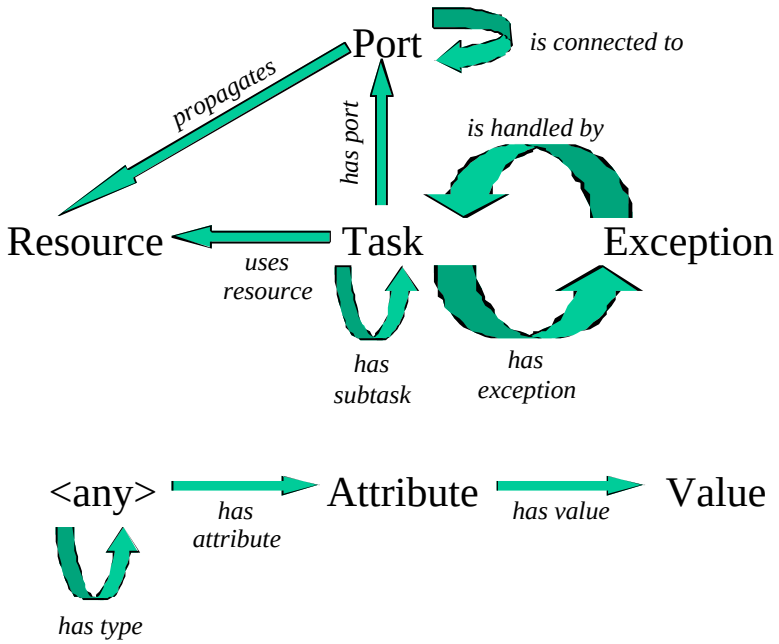

Figure 3: Process model formalism.

The key components of this representation include:

- *Attributes:* Processes can be annotated with attributes that capture such information as a textual description, typical performance values (e.g. how long a process takes to execute), and so on.
- *Decomposition:* A process can be modeled as a collection of processes that can in turn be broken down ("decomposed") into sub-processes.
- *Resource Flows:* All process steps can have input and output *ports* through which *resources* flow. One innovation we use is to recognize that processes can be divided into 'core' activities as well as those involved in coordinating the flow of resources between core activities [11]. This insight allows us to abstract away details about how sub-processes coordinate with each other, allowing more compact service descriptions without sacrificing significant content.
- *Mechanisms:* Processes can be annotated with the resources they *use* (as opposed to consume or produce). For example, the Internet can serve as a mechanism for a process.
- *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 [12].

Let us consider a simple example to help make this more concrete (Figure 4):

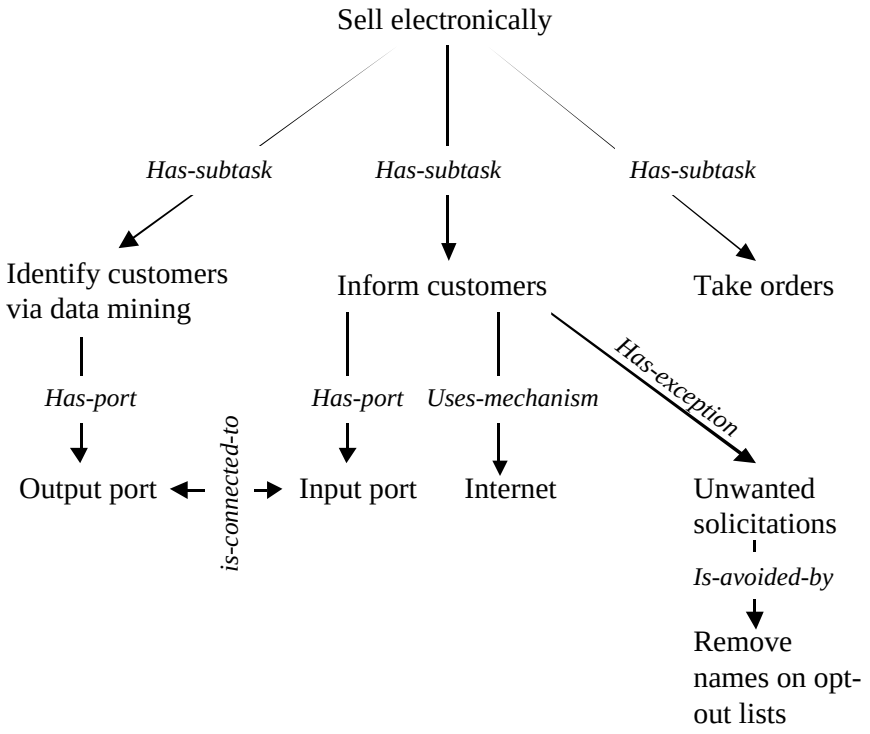

Fig. 4. An example of a process-based service model

This represents the process model for a service for selling items electronically. The plain text items represent entities (such as exceptions, ports and tasks), while the italicized items represent relationships between these entities. The substeps in this service model include ‘identify potential customers via data mining’, ‘inform customers’ (which uses the Internet as a mechanism), and ‘take orders’. The potential exception of sending out unwanted solicitations is avoided by filtering out the names of individuals who have placed their names on ‘opt-out’ lists. Each of the entities can have attributes (not shown) that include their name, description, and so on.

This representation is equivalent to other full-fledged process modeling languages (e.g. IDEF [13], PIF [14] and PSL [15]) with the exception that it does not currently include primitives oriented at expressing control semantics, i.e. that describe *when* each subtask gets enacted. Such primitives were excluded for two reasons. One is that the bulk of the variation between different process modeling languages occurs in the realm of representing control semantics, and we wanted to begin with a formalism that mapped directly to a wide range of existing process models. The other reason is that it is our experience to date that most service queries are concerned with *what* a process does, rather than *when* the parts of the process gets enacted.

Modeling service behaviors as process models may of course involve some manual effort, but we argue that this need not be a significant barrier to adoption of this approach. In our experience

even relatively simple process models can enable qualitative increases in retrieval precision. Because process formalisms are so widely used, many services will already have process models defined for them. Software applications and business models, for example, are routinely described using flow charts or other process modeling formalisms. Providers in competitive marketplaces will be motivated to create such models as a way of making the advantages of their services explicit. Someone offering an electronic sales service may, for example, wish to capture their ability to handle unwanted solicitations or use data-mining in their published models. Capturing service semantics enables, in addition, many important uses other than retrieval, including service execution, interoperation, composition and execution monitoring [16]. Since creating a service represents a substantial effort, it is unlikely that the relatively small increment needed to model that service as a process will be a major concern. Finally, template-based modeling (see 'future work') can be used to substantially reduce the modeling effort involved.

Defining Queries*:* We have defined a query language called PQL (the Process Query Language) designed for retrieving process models (see [Berstein, 2002 #4272] for a formal description). Process models can be straightforwardly viewed as entity-relationship diagrams made up of entities like tasks characterized by attributes and connected by relationships like 'has-subtask'. PQL queries are built up as combinations of five clause types that check for these elements:

· Entity <entity> isa <entity type>
· Relation <source entity> <relationship type> <target entity> [*]
· Attribute <attribute> of <entity> {includes | equals} <value>

The 'entity' clause matches any entity of a given type (the entity types include task, resource, port and so on). The 'relation' clause matches any relationship of a given type between two entities (the relationship types include has-subtask, has-specialization, has-port, and so on). The optional asterisk finds the transitive closure of this relationship. The 'attribute' clause looks for entities with attributes that have given values. 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.

We have also found it useful to include an operator for grouping clauses into sub-queries:

· When {exists | does-not-exist} <query>

The 'when' clause enables further pruning by submitting the matches returned by a sub-query to a predicate. At present, only two built-in predicates have been defined but this will likely expand.

Let us consider a simple example to help make this more concrete. The query below searches for a sales service that uses the Internet to inform customers:

```
attribute "Name" of ?sell includes "sell"
when exists (relation ?process has-subtask ?subtask *
                attribute "Name" of ?subtask includes "inform"
                attribute "Description" of ?subtask includes “internet")
```

The first clause searches for a processes in the ontology whose name includes “sell”, and the second checks if any subtasks of these services are “inform” processes with “internet” in their description. A PQL query is thus equivalent to a sub-graph pattern, and any search for a process model can then be treated as finding the nodes of type *task* which match the graph pattern that represents the query.

Finding 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 bindings that survive to the end represent the matching services. Query performance can be increased by using such well-known techniques as query clause re-ordering. This can be viewed as an application of existing work on graph grammars [17]. Graph grammars model artifacts as graphs and manipulating them using rules that rewrite segments of these graphs. We can in principle implement service retrieval as a graph grammar rule that notifies a user of all matching instances found in a database of process models. The unique contribution of our work lies in identifying how we can exploit the particular semantics of process models to enable retrieval-specific capabilities, such as semantics-preserving query mutation and automated service indexing (see ‘future work’ below), that would be difficult or impossible to achieve without knowledge of the link and node semantics.

4. How Well Does it Work?

A PQL interpreter has been implemented, in Common Lisp. We have defined and enacted many PQL queries and our experience is that PQL can be used in a straightforward way to capture queries drawn from many domains. We have also found that PQL enables qualitative improvements in retrieval precision, even with quite simple process models. Our test case involved searching for services that sell products using the Internet to inform customers. The Process Handbook database was used as our service repository. All queries had perfect recall. The keyword based search (“sell” AND “internet” AND “inform”) had a retrieval precision of 5%. We compared this to a range of PQL queries that made increasing use of the information that can be encoded in process-based service models.

Query	Precision
attribute "Name" of ?service includes "sell" when exists (relation ?service has-subtask ?subtask * attribute “Name” of ?subtask includes "inform" attribute ?attr of ?subtask includes “internet")	72%
attribute "Name" of ?service includes "sell" when exists (relation ?service has-subtask ?subtask * attribute “Name” of ?subtask includes "inform" **relation ?subtask uses-mechanism ?mechanism** **attribute “Name” of ?mechanism includes “Internet”)**	81%
attribute "Name" of ?service includes "sell" when exists (relation ?service has-subtask ?subtask * attribute “Name” of ?subtask includes "inform" relation ?subtask uses-mechanism ?mechanism attribute “Name” of ?mechanism includes “Internet” **attribute “Name” of ?class equals “Inform”** **relation ?class has-specialization ?subtask)**	100%

The first query made use only of the task decomposition information in the process models. This allowed us to avoid false positives wherein, for example, the Internet was used but not for the task of informing customers. Even this basic level of process modeling resulted in qualitatively higher precision of 72%. The second query added the use of task mechanism information, retrieving only processes wherein the Internet is used as a mechanism for the inform subtask, eliminating false positives where the term “Internet” was merely mentioned in the description somehow. This increased retrieval precision to 82%. The final PQL query added the feature of exploiting the Handbook process taxonomy to only retrieve processes with subtasks that are a specialization of the generic “inform” process. This brought retrieval precision up to 100%. This final example shows that PQL can be integrated with other retrieval approaches (in this case, concept-based retrieval) to produce higher retrieval precision than is possible using either approach individually.

Another important result concerns the computational complexity of PQL queries. Even though our experience with the prototype implementation has been favorable (i.e., queries generally take several seconds at most, even though our implementation does not exploit well-known query optimization techniques), it is important to evaluate how performance will scale with the size of the service model database. We were able to show that PQL is the equivalent of a DATALOG-type language whose computational complexity is polynomial. This is a good result: polynomial

complexity implies that the computation needed to enact a PQL query scales comparably with that of widely-accepted retrieval technologies such as SQL.

Further details on these evaluations, both empirical and analytic, are available in [18].

5. Next Steps

While our results are promising, important challenges remain. One key issue involves *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, use different keywords to mean the same thing, or include a given subtask several levels down the process decomposition, while in the query that subtask may be just one level down. In order to avoid poor recall we must therefore provide a 'fuzzy' or imprecise retrieval scheme that is tolerant of such differences. We are exploring, for this purpose, the use of standard synonym-matching techniques, as well as the notion of semantics-preserving query mutation that modifies a service query to produce a whole space of semantically similar variants.

Another key issue concerns *rapid service modeling*. As pointed out earlier, many services already have process models created for them as standard practice, so we can simply translate these models, often automatically, into service descriptions suitable for retrieval by PQL. In the absence of such pre-existing models, manual service modeling can, we believe, be substantially sped-up through the use of templates. We simply find an existing description for a service that is similar to ours, and define our own service description as a modification thereof. The more service providers that have created process-based descriptions, the more likely it is that a suitable template will be available to speed our own service description efforts. We can reasonably expect that research groups, individual enterprises and consortia will define and market industry-specific template libraries. The MIT Process Handbook, for example, has already created a business process repository with over 5000 process descriptions ranging over such areas as supply chain logistics, hiring, and so on [19]. This project has developed sophisticated tools for process modeling that allow a knowledgeable user to create new models in a matter of minutes.

We are currently engaged in empirically *evaluating* PQL by asking roughly sixty users to define a database of services, a suite of keyword, table and process-based queries, and a listing of the correct service-query matches. Using this data, we will be able to assess the relative precision and recall of these different retrieval techniques.

Finally, we plan to address the issue of defining a *more accessible interface* for human users of PQL. Creating PQL queries, as with many query languages, requires some technical expertise. Possible directions include developing graphical or natural language front-ends for PQL queries.

6. Contributions

High retrieval service precision is widely recognised as a critical enabler for important uses that range from finding useful software components or applications, to uncovering relevant best practice models, to tracking down people or organisations with the skills you need. Our work can be viewed as representing a new class of service retrieval technology that helps achieve these goals (Figure 5):

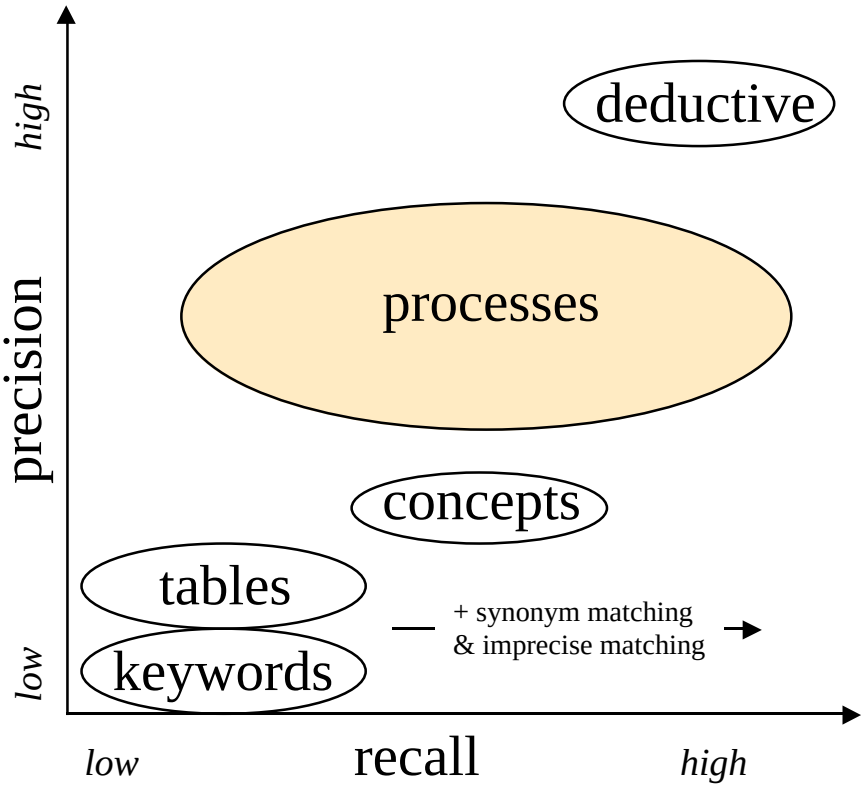

Figure 5. The contribution of process-based service retrieval technology.

Our evaluations suggest that process-based queries produce retrieval precision qualitatively greater than that of existing service retrieval approaches, while retaining acceptable complexity for query enactment. See http://ccs.mit.edu/klein/ for further details on this work.

7. Acknowledgements

This work was supported by the Army Research Laboratory of the US Army Research, Development and Engineering Command. I would like to thank Dr. Dana Ulery, Director of the ARL Knowledge Fusion Center, for supporting and helping to identify potential applications for this work.

8. References

1. Bakos, J.Y. (1997). *Reducing Buyer Search Costs: Implications for Electronic Marketplaces.* Management Science. **43**.
2. Magnini, B. (1999). *Use of a lexical knowledge base for information access systems.* International Journal of Theoretical & Applied Issues in Specialized Communication. **5**(2): p. 203-228.
3. Lindig, C. (1995). *Concept-Based Component Retrieval.* In the proceedings of International Joint Conference on Artificial Intelligence.
4. Diamantini, C. and M. Panti (1999). *A conceptual indexing method for content-based retrieval*. In the proceedings of Tenth International Workshop on Database and Expert Systems Applications. p. 192 -197.
5. Khan, L. and F. Luo (2002). *Ontology construction for information selection*. In the proceedings of 14th IEEE International Conference on Tools with Artificial Intelligence (ICTAI 2002). p. 122 -127.
6. Alsaffar, A.H., et al. (2000). *Enhancing Concept-Based Retrieval Based on Minimal Term Sets.* Journal of Intelligent Information Systems. **14**(2-3): p. 155-173.
7. ESPEAK, *Hewlett Packard's Service Framework Specification.* 2000, HP Inc.
8. Richard, G.G. (2000). *Service advertisement and discovery: enabling universal device cooperation.* IEEE Internet Computing. **4**(5): p. 18-26.
9. Kuokka, D.R. and L.T. Harada (1996). *Issues and extensions for information matchmaking protocols.* International Journal of Cooperative Information Systems. **5**: p. 2-3.
10. Meggendorfer, S. and P. Manhart (1991). *A Knowledge And Deduction Based Software Retrieval Tool*. In the proceedings of 6th Annual Knowledge-Based Software Engineering Conference. IEEE Press. p. 127 - 133.
11. Malone, T.W. and K. Crowston (1994). *The interdisciplinary study of coordination.* ACM Computing Surveys. **26**(1): p. 87-119.
12. Klein, M. and C. Dellarocas (2000). *A Knowledge-Based Approach to Handling Exceptions in Workflow Systems.* Journal of Computer-Supported Collaborative Work. Special Issue on Adaptive Workflow Systems. **9**(3/4).
13. NIST, *Integrated Definition for Function Modeling (IDEF0).* 1993, National Institute of Standards and Technology.

14. Lee, J. and M. Gruninger (1998). *The Process Interchange Format "The Process Interchange Format and Framework v.1.2.* Knowledge Engineering Review. (March).
15. Schlenoff, C., et al. (1999). *The essence of the process specification language.* Transactions of the Society for Computer Simulation. **16**(4): p. 204-16.
16. Narayanan, S. and S. McIlraith (2002). *Simulation, Verification and Automated Composition of Web Services*. In the proceedings of Eleventh International World Wide Web Conference (WWW-11). Honolulu, Hawaii.
17. Rozenberg, G., ed. (1999). *Handbook of Graph Grammars and Computing by Graph Transformation*. Vol. 3. World Scientific.
18. Bernstein, A. and M. Klein (2002). *Towards High-Precision Service Retrieval*. In the proceedings of The International Semantic Web Conference (ISWC-02). Sardinia Italy.
19. Malone, T.W., et al. (1999). *Tools for inventing organizations: Toward a handbook of organizational processes.* Management Science. **45**(3): p. 425-443.