

Center for

eBusiness@MIT

<http://ebusiness.mit.edu>



A research and education initiative at the MIT Sloan School of Management

Executive Summary:
Improved Process Retrieval Using Process Ontologies

Paper 147

**Mark Klein
Abraham Bernstein**

January 2002

For more information,

please visit our website at <http://ebusiness.mit.edu>

or contact the Center directly at ebusiness@mit.edu or 617-253-7054



Executive Summary:

Improved Process Retrieval Using Process Ontologies

Mark Klein

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

Abraham Bernstein

Stern School of Management
New York University
bernstein@stern.nyu.edu

The Challenge

Increasingly, on-line repositories such as the World Wide Web are being called upon to provide access not just to static documents that collect useful *information*, but also to *services* that provide useful *behavior*, such as on-line software applications, software component libraries, process models, and individuals or organizations who can perform particular functions. Better tools are needed 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.

Our Approach

Our central claim is that these goals can be achieved through the sophisticated use of process ontologies. We begin with the MIT Process Handbook, a process ontology editing tool and associated knowledge base of over 5000 business processes. We index a service by placing a process model describing the service, plus all of its components (subtasks, input and output ports, and so on) into the proper places in the ontology. A query is also defined as a typed process model. A straightforward algorithm can be used to find matches between the query and indexed services.

Benefits

Increased Precision: Process models organized into a process ontology capture service behavior semantics more completely than current (mainly keyword-based approaches), resulting in 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 principled use of semantics-preserving query mutation operators.

Easy to index services: [Semi-]automated classification of flowchart-based service models should substantially reduce the manual service modeling burden, perhaps eliminating it completely.

Improved Service Retrieval Using Process Ontologies

Mark Klein

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

Abraham Bernstein

Stern School of Management
New York University
bernstein@stern.nyu.edu

The Challenge

Increasingly, on-line repositories such as the World Wide Web are being 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 be invoked remotely by people or software
- ◆ *Software components* that can be downloaded for use when creating a new application
- ◆ *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 white paper describes a promising line of work, based on the sophisticated use of process ontologies, for creating improved service retrieval technologies.

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):

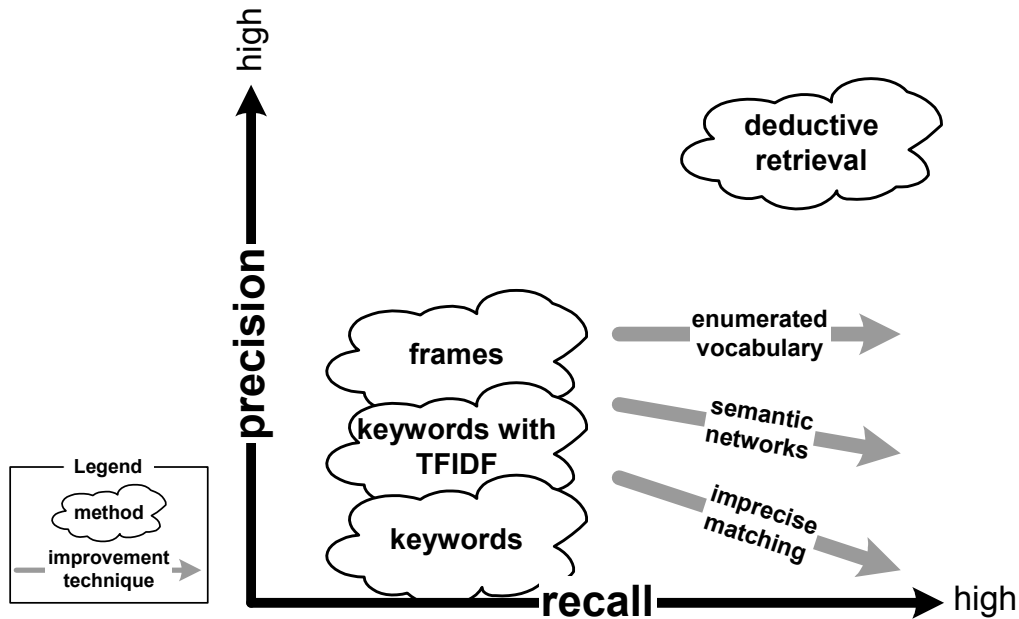


Figure 1. The state of the art in service retrieval.

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.

Several techniques have been developed to address these problems. One is to require that items and queries are 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 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 [5] but this potentially can reduce precision. Search engines like google.com [6] 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 achieve fairly high recall but 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 [7] [8] [9] [10] [11] 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:

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.

Both items and queries are described using frames: 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, Salutation, UDDI [12]) 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 [13]) 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 (hypernym) 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.

The frame-based approach is taken one step further in the *deductive retrieval* approach [14] [15] [16] 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 [14]

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 2 is non-trivial to formalize correctly: imagine trying to formally model the behavior of Microsoft Word or an accounting package! This problem is exacerbated by the fact that, when so modeling a service, one needs 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. A complete formalization of the service would need to capture both these elements. The second difficulty is that the proof process implicit in this kind of search can have a high computational complexity, making it extremely slow [14]. Our belief is that these limitations, especially the first one, make deductive retrieval unrealistic as a scalable general purpose service search approach.

Other approaches exist, with specialized application areas. One is execution-based retrieval, wherein software components are selected by comparing their actual I/O behavior with the desired I/O behavior. This approach is suitable only for contexts where observing a few selected samples of I/O behavior are sufficient to prune service set [17] [18] [19].

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.* Services can be indexed (to facilitate their subsequent retrieval) by placing them and their components into the appropriate sections of the ontology. Queries can be expressed as process model templates. The matching algorithm can find all the services whose process models match the query, using the semantic relationships encoded in the process ontology, to enable increased precision without sacrificing recall. Our approach can thus be viewed as having the following functional architecture:

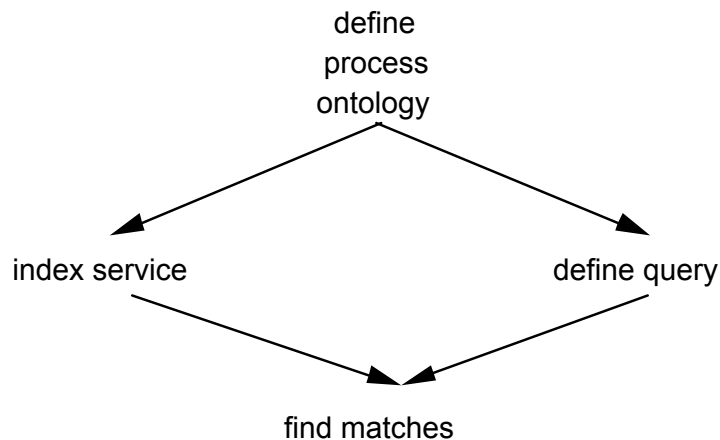


Figure 3. Functional architecture of our proposed service retrieval technology.

Define Process Ontology

Our approach differs from previous efforts in that it is based on full-fledged 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 [20] [21]. 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). The handbook tools include a Windows-based tool for editing the Handbook database as well as a Web-based tool for read-only access. 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 address.

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

Dependencies: Another key concept we use is that coordination can be viewed as the management of *dependencies* between activities [20]. 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 [22].

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.

The process representation provided by the Handbook is equivalent in formal expressiveness to other full-fledged process modeling languages (e.g. IDEF [23], PIF [24], PSL [25] or CIMOSA [26]) and more powerful than the frame-based languages used in previous service retrieval efforts, by virtue of adding the important concepts of resource dependencies, task decompositions, and exceptions.

Index Services

Services are indexed into the process ontology so that they may be retrieved readily later on. Indexing a service in our approach is identical to adding any new process to the ontology: it 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 Handbook project has developed sophisticated tools for process ontology editing that have been refined over nearly eight years of daily use. Using these tools, most process models can be classified by a knowledgeable user in a matter of minutes.

We can take this one important step further, however. 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 [27] [28] [29]
- Manually developed frame-based models of the service [8]

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 propose to improve on these approaches by developing tools for [semi-]automated classification of flowchart-like process representations. Such flowcharts 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 flowchart models defined for them. These flowcharts may even in some cases already be classified into some kind of process ontology. The greater expressiveness of flowcharts as compared to standard frame-based service models (flowcharts

often include task decomposition and resource flow information, for example) 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. This approach can be optimized by using machine learning algorithms to refine the weights of these sub-measures based on classification accuracy, which we can assess for example by observing how closely the algorithm places processes to their original position in the Handbook process ontology. 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.

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 [30]. Our own experience with the Process Handbook suggests however that browsing can be slow and difficult for all except the most experienced users. 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 [31]. 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 clauses that check for given entities and relationships:

<entity> isa <entity type> :test <predicate>

<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 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 to make this more concrete. The query:

```
?proc isa sales-process
?proc has-subtask ?sub
?sub isa internet-process
```

searches for a service that performs a sales function where at least one subtask uses the internet.
The query:

```
?proc isa fast-fourier-transform-process
?proc has-port ?port1
?port1 isa input-port
?port1 propagates-resource ?res
?res isa digitized-signal
?res has-attribute ?attr
?attr isa size
?attr has-value ?val :test (> ?val 1000)
```

searches for a fast fourier transform service that can accept input with more than 1000 time points in it.

PQL, like any query language, is fairly verbose and requires that users be familiar with its syntax. An important part of the proposed work will be to develop more intuitive interfaces suitable for human users. One possibility exploits the process models in 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 the desired service behavior via “mix-and-match”, i.e. as a combination and refinement of existing elements in the process ontology. A user could for example specialize the “fast fourier transform” process in the ontology (found for example using simple keyword search) to include the constraint “> 1000” on the model size attribute. Another approach is to allow users to enter queries using restricted subset of English (an approach that has been applied successfully to traditional database access), so the second query above for example could be expressed as:

Find a “fast fourier transform” whose input “size” can be greater than 1000

These two approaches can of course be combined, so that for example a simple natural language query returns some candidate classes that are selected from and refined to define the final, more complete, query. Finally, users can define their desired service using a flowchart model and use the automated classification scheme described above to map this to a PQL query.

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 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. While we have not yet evaluated PQL’s performance in detail yet, it is clear that its primitive query elements have low computational complexity.

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 terminology used to describe a task may differ between the query and service model. 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 [32], 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 propose to explore for this purpose the use of query transformation 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:

1. allow a type specification to be more general
2. allow a subtask to be any number of levels down the task decomposition hierarchy
3. allow ‘siblings’ or ‘cousins’ of a task to constitute a match
4. relax the constraints on a parameter value

We can use these operators to produce a whole space of queries representing plausible variants of the original query. For example, we can take the query we presented above (“find a fast fourier transform that can handle inputs with 1000 points”) and use the transformation operators to produce such variants as:

- ◆ find a fast fourier transform that can handle at least 800 points (rule 4)
- ◆ find a spectral analysis service (rule 1)

Similarly, we can take the query “find a sales process with a subtask that uses the internet” and use transformation operators to produce such variants as:

- ◆ find an exchange process with a subtask that uses the internet (rule 1)
- ◆ find a sales process where any subtask (at any depth in the task decomposition) uses the internet (rule 2)
- ◆ find a sales process that uses EDI (rule 3)

The results that result from submitting these queries can be ordered using the semantic similarity metric described above, in order to produce the reduced sensitivity to modeling differences that we need.

Work Plan

Year 1 of our project will focus on developing and evaluating base versions of the core search engine technology, including the PQL query language, matching algorithms, and imprecise query mutation techniques. Our evaluation will focus on speed, precision and recall, comparing our approach to existing technologies including commercial search engines. There are many possibilities for getting the on-line service and query examples we will need to do these tests, for example from the query logs commercial information service providers (e.g., Northern Light, Reuters or eSpeak).

Year 2 will focus on refining these search technologies as well as developing and evaluating techniques to reduce the service and query modeling burden for human users. These will include the advanced UIs and [semi-]automated service classification described above. The query/service definition burden can be assessed by timing human subjects.

Year 3 will address developing more advanced capabilities (such as automatic refinement of PQL queries based on relevance feedback, and the use of user and domain models to speed query definition) in addition to demonstrating the utility of this approach in a range of domains such as retrieving software from a component library, finding human experts or organizations with given skills, matchmaking in agent-based systems, retrieving business process models from best practice libraries, and so on.

We do not plan to explore scalability issues because we believe that existing technologies developed for other distributed database applications will be directly applicable to enabling the replicated architecture necessary for scaling up this approach.

We propose to disseminate our results to the interested academic communities through publications, conference presentations and tutorials. We anticipate that this technology, if successful, will have substantial commercialization potential, for example by web portals and software engineering tool vendors.

Benefits

Our proposed project will result, will believe, in service retrieval technology that makes the following important contributions:

Increased Precision: Our approach differs from previous efforts in that it models services using full-fledged process models organized into a process ontology. Such models capture service behavior semantics more completely than keywords or frames by virtue of representing task decompositions, resource flows, exceptions, and so on. 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 principled use of semantics-preserving query mutation operators coupled with our process semantics similarity metric.

Easy to define queries: Existing techniques from the Handbook project coupled with our proposed advances in “mix-and-match” as well as natural language query definition should result we believe in an acceptable increase in the query definition burden when traded off against the increase in retrieval precision.

Easy to index services: [Semi-]automated classification of flowchart-based service models, exploiting the increased accuracy of our semantic process similarity metric, should substantially reduce the manual service modeling burden, perhaps eliminating it completely.

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. Nejme, *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.