Proceedings of DETC'03
ASME 2003 Design Engineering Technical Conferences and
Computers and Information in Engineering Conference
Chicago, Illinois, USA, September 2-6, 2003

DETC2003/CIE-48219

EXPERIENCES IN DESIGNING HIGHLY ADAPTABLE EXPERTISE FINDER SYSTEMS

Gareth Hughes and Richard Crowder
Intelligence, Agents, Multimedia Group
Department of Electronics and Computer Science
University of Southampton
Southampton
{gvh,rmc}@ecs.soton.ac.uk

ABSTRACT

A strategy or method for identifying an expert is at the core of any Expertise Finder system. The strategy used must reflect the social requirements of the organisation and its assets, culture and technology. An Expertise Finder strategy must attempt to emulate the steps a member of that organisation would do to find the answer to their questions. Our collaboration with a large manufacturing enterprise has helped us evolve an approach to building Expertise Finder systems. Our approach contains the required flexibility to cope with the mix of technological, social and legacy factors that make each deployment of an Expertise Finder system very different to any other. The component based design allows a deployer to customise the basic system to utilise their relevant technical resources and then to build a unique strategy algorithm to produce a valid set of recommendations. The design produces challenging questions of how to produce a procedure for designing, deploying and evaluating the Expertise Finder strategy. This paper discusses the evolution of this design strategy and summarises our experiences to date.

Keyword: Expertise Finding, Knowledge Based System, Human Resources, Agents

Introduction

In the course of many engineering activities, people face problems that they cannot solve independently. Their natural response is to study past experiences, and re-use previously acquired knowledge, either from their own experiences or from resources within their organisation. It has been estimated that 90% of industrial design activity is based on variant design [1], while during a redesign activity up to 70% of the information is taken from previous solutions, [2].

In many cases, access to documentation through hypermedia or similar document management systems will be adequate [3]. However to solve many problems people need to have access to specific expertise. In this paper, the term expertise assumes the embodiment of knowledge and skills within individuals. This definition distinguishes expertise from an expert, as an individual may have different levels of expertise about different topics. Expertise can be topical or procedural and is normally stored and valued within any organisation. In some cases expertise can be captured from a person and used to populate a specific database which works very well if the problem is restricted to a very specific domain, for example robot maintenance, [4] key personnel (managers, senior employees, information concierges) [5], facilitating the contacts. With the advent of knowledge management systems it is possible to automate the process through Recommender Systems or Expertise Finders (EF). A recommender system that suggests people who have some expertise with a problem holds the promise to provide, in a small way, a service similar to these key personnel. Expertise recommender systems can also reduce the load on people in these roles and provide alternative recommendations when these people are unavailable.

It could be argued that there can be no single definition of an expert. It is completely dependent on the context of the query. This is governed by the subject, the organisational setting and the current activity that the person asking the question is undertaking. This poses serious difficulties for those attempting to design software to locate experts or expertise. Software needs to be configured to the organisation's social and technical structure. This is complicated and expensive to do. If one design approach will not 'fit all'then the software needs to be highly flexible to be reusable across a number of enterprises. This paper describes the evolution of our ideas and designs for a flexible EF design. We describe our previous work and show how the experiences with this system generated a set of requirements for a better EF. The implementation of this new design is covered in detail.

In the recommendations provided by the EF, trust is important, this can be achieved by showing why people were not recommended or why a document was not considered so important. A document might seem relevant based on a full text search but is actually twenty years old, an important factor in some situations, but not in others. The provision of evidence for its decisions in the form of a list of documents and other data is considered a key EF output.

Our approach to EF contrasts with a number of reported systems where web based information is used to provide the recommendation [6, 7, 8]. Answer Garden 2 [9] has an explicit expertise-location engine and provided computer-mediated communications mechanisms to find others with a range of expertise, though the mechanisms were not very elaborate. A different approach was taken by McDonnald [5] who used software developed by employees to identify their expertise in various aspects of software development.

System Rationale

When attempting to find an answer to a query people will tend to use the social network around them. It is natural to first ask people nearby if they know the answer or if they can recommend someone else who may know the answer. Thus a chain of connections are made utilising the experienced members of an organisation. As people are now being moved around organisations at a faster rate and organisations are becoming increasingly distributed this model will fail. There may be no social connection between geographically separated groups even though they work on similar problems. Our system attempts to alleviate this by using the company's own resources to recommend people to contact. It does not replace the social network but attempts to speed up the connection making process.

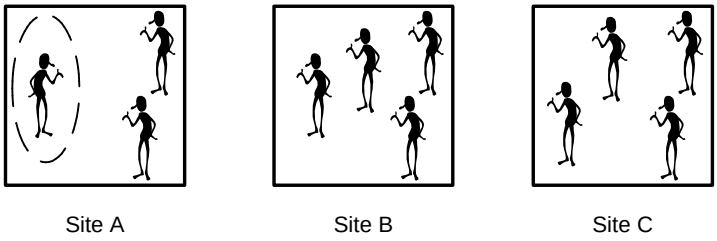
The problem that is being addressed is summarised in Fig. 1(a), how does a person located in Site A, locate the best expertise to solve a specific problem? The person's local network will only extend to within the site, and therefore expertise in other sites can not readily accessed, and in this example it proves impossible to locate a local expert. It should be remembered that sites can share common problems, but not necessarily be easily accessible to each other. While the sites may not form a cohesive social network, they do share common sets of resources, including e-mail, phone books, publication and report repositories, Fig. 1(b). In our approach to EF, these information repositories are used to identify the required expert, Fig 1(c). Our experience has shown that no two organisations are the same, hence any methodology to find an expert is highly specific to an organisation.

In principle this is acceptable, however when a person wishes to contact an *expert*, there are additional social factors that need to be taken into account. Without these factors, the single expert will be swamped with queries for everyone across the organisation. The appropriate person will depend on the query and the users requirements, typically the peer-to-peer approach is considered best in the first instance, however the person requiring the expertise needs to be free to make a valued judgement as whom to approach. It is for this reason we make available all the sources for review. As discussed by McDonald [10] the details matter in successful expertise location. The heuristics used to select the expert are bound to the organisational environment. Systems that augment expertise locating must be capable of handling large number of details that depend on the specific context and problem.

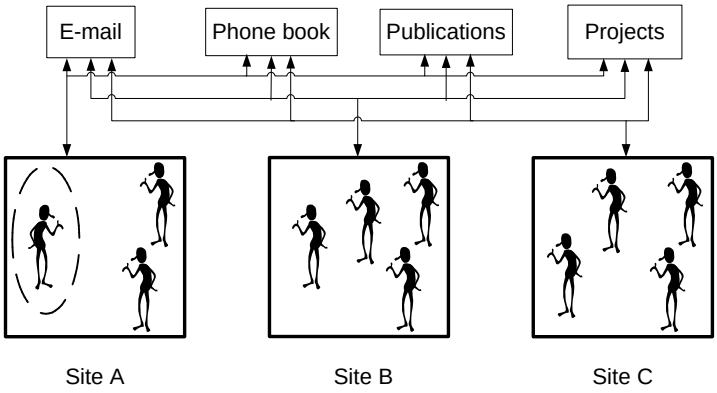
The strategy or method for identifying an expert is at the core of any EF system. This strategy must reflect the social requirements of the organisation and its assets, culture, technology. A strategy must attempt to emulate what a member of that organisation would do to find the answer to their questions. Whether it is using resources or by talking to people around them. The social network they have built up will play a key role in that strategy. However if their social network is constrained, for instance by their relative inexperience and hence small network, or by only knowing the people at their outpost within a geographically distributed company, then they will need to have that network boosted by technology. Technology can help but, we believe, should not attempt to replace such social networking. The goal of our research is not to provide answers to deeply technical questions by providing a search portal but to help the user find someone who would be a good candidate to ask and back that up with recommended documents. By aiming to use large-scale resources we aim to provide company wide social connections.

The systems we have built use the resources already existing within an organisation. The document repositories, the phone book listings, web sites and so forth. We have created distributed systems to tap into these resources and combine their results in some fashion to provide an overall result. In the examples we have worked on the amount and type of online resources and legacy data varies considerably and we have come to realise that there can be no assumptions made about what will be available.

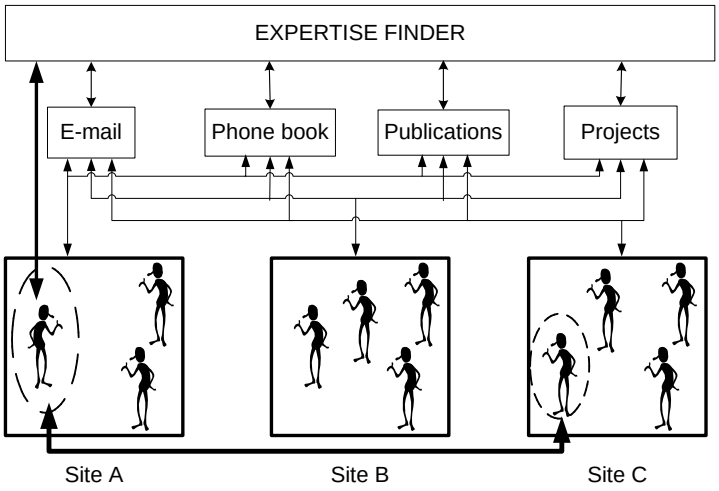
In the systems we have developed we have broken down the



(a) Local social network fails to find an expert.



(b) Individuals are known to the organisation electronic resources



(c) Expertise Finder identifies an expert regardless of location

Figure 1. Overview of the Expertise Finder concept

process of searching for an expert into a number of steps. The system is designed with the assumption that all of the components will need to be changed at some time in order to adapt the framework for use in a new situation.

People who need to solve problems during their work have two main ways to find a solution. The first is to look at documents of some form, the other is to ask other people. Both methods will lead them on a trail of discovery through their organisation until they eventually find a useful answer or give up. Knowing where to look and who best to ask are the core problems of finding expertise. In some organisations it may be that the archives will be a sufficient source of information. More often it is the ex-

perience of knowledge workers which will be more important. Electronic systems that attempt to aid a person in their quest for the answer draw together the various sources of documents and records within an organisation as well as providing knowledge about population. From these resources it is hoped that a strategy can be found to aid a person find documents or people to help them with their answer.

Overview of our initial Expertise Finder

Our first implementation of an Expertise Finder [11], consisted of a number of DIM (Distributed Information Management) Agents operating within SoFAR (Southampton Framework for Agent Research) [12].

In order to communicate with each other agents use a shared understanding of a domain called an ontology. Ontologies provide a mechanism to allow communication and interaction about a real world domain [13]. Pragmatically it allowed us to concentrate on high level concepts rather than spend time on the implementation details such as communications and data representation. It therefore follows that the design of the ontology is crucial to the project and careful work is required to correctly understand and map the real world situation into the ontological vocabulary. Further technical details of how ontologies are implemented and used in the SoFAR framework can be found in [12]. The ontologies used in this work were designed previously but extended here. They represent the activities and people in our research group. A detailed explanation of their design and implementation can be found in [14].

The original agent based EF system consisted of a main agent, the EF Agent, which used a set of simpler *Source Agents* in some algorithm to determine a list of people and documents to recommend an expert to the user, Figure 2.

Source Agents (the Publications and Directory Services agents) were designed to represent sources of information and data within the organisation. These could range from the simple, an agent that understands the data stored in the internal phone book, to more complex knowledge such as an agent interface to a publications database. For instance the Directory Services agent could answer queries about the location of people or return all of the people with a certain phone number.

The procedure involved in this system seems to be simple at first glance but the complexities of implementation taught us many lessons. The agents were not as independent of each other as was planned. The complex interactions between the source agents and the large amounts of error checking that needed to be undertaken was a classic example of writing systems to deal with real-life data. Also if one of the underlying data sources changed significantly then it may require an ontology modification and hence the EF Agent will need to be modified. This demonstrates that where complex queries and interactions between component systems are required there will be more brittleness to the design.

Even though we used agents and ontologies to help mask implementation details of underlying resources the agents were still highly interconnected. These interconnections were often the result of legacy data or 'dirty data'and the agent framework made it much harder to deal with these issues. It was realised that even though a well designed EF system should be reusable there would need to be a significant component developed for each specific deployment. This would house the custom steps to reflect the local ways of finding an expert and cope with the problems found in the legacy systems. This leads to a requirement that the developer must understand the relationships between the data that is being worked with as well as the clients needs.

It was our conclusion that this approach came at the problem from the wrong direction, while the agent approach was flexible, the system was not geared towards being able to interchange components so readily and the necessity of designing agents and ontologies was not strictly of benefit in this case. In particular the management of the ontology, was and could become a major limiting factor, as every new application or change of data sources required extensive modifications to the ontological structure.

The Revised Expertise Finder

The difficulties we faced in improving the original EF system led to a new design emerging. The system was implemented using Java Servlets and is interfaced via a simple search-form web page. The user enters a query and the system will return a ranked list of documents plus a ranked list of people to contact. On entering a technical query the system's goal is to find documents and people pertinent to that query according to some predefined strategy. Fig. 3 shows the system architecture of our current implementation of the Expertise Finder.

When the system is invoked by a users submission a certain *expertise finding strategy* will be activated. This is a piece of code that decides which *components* to use and how to manipulate the results they return. The strategy will employ various other components in a sequence. The first type of components are called *generators*. These perform the search and will produce lists of documents or people. The second set of components are termed *scorers*. These will take the existing results and give them new scores based on on a predefined algorithm. The *strategy* keeps all of the scores that each *component* produces. It then computes a final score for each document or person. This is done using a ratings system for each of the scores. The ratings can come from the *strategy* code itself or can make use of an individual's preferred ratings.

Fig. 4 and Fig. 5 show the final results of a query, in the current implementation these are shown on the same web page. The web page shows a ranked list of documents followed by a ranked list of people. Each is given a score that is represented by a horizontal bar graph. The score is the result of a series of queries to find and score documents and people from various sources. The

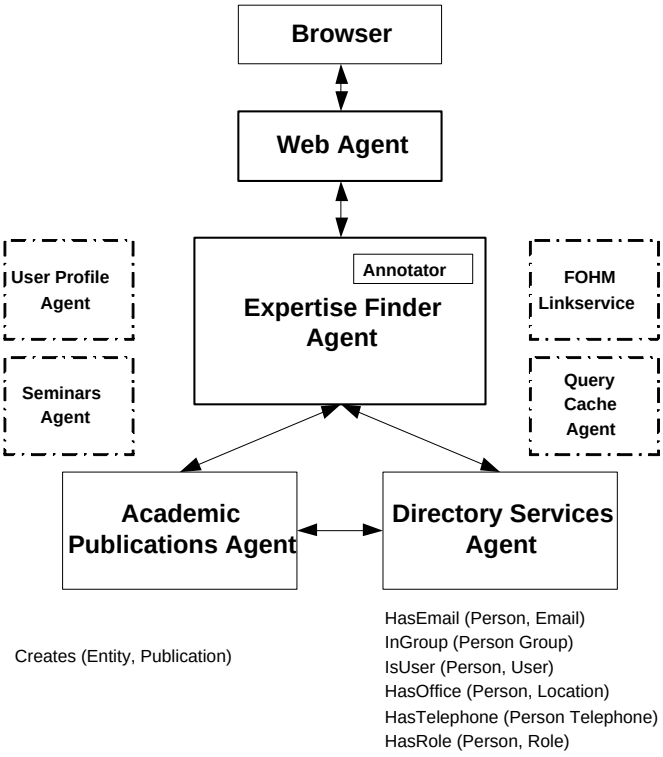


Figure 2. The architecture of our earlier agent based expertise finder.

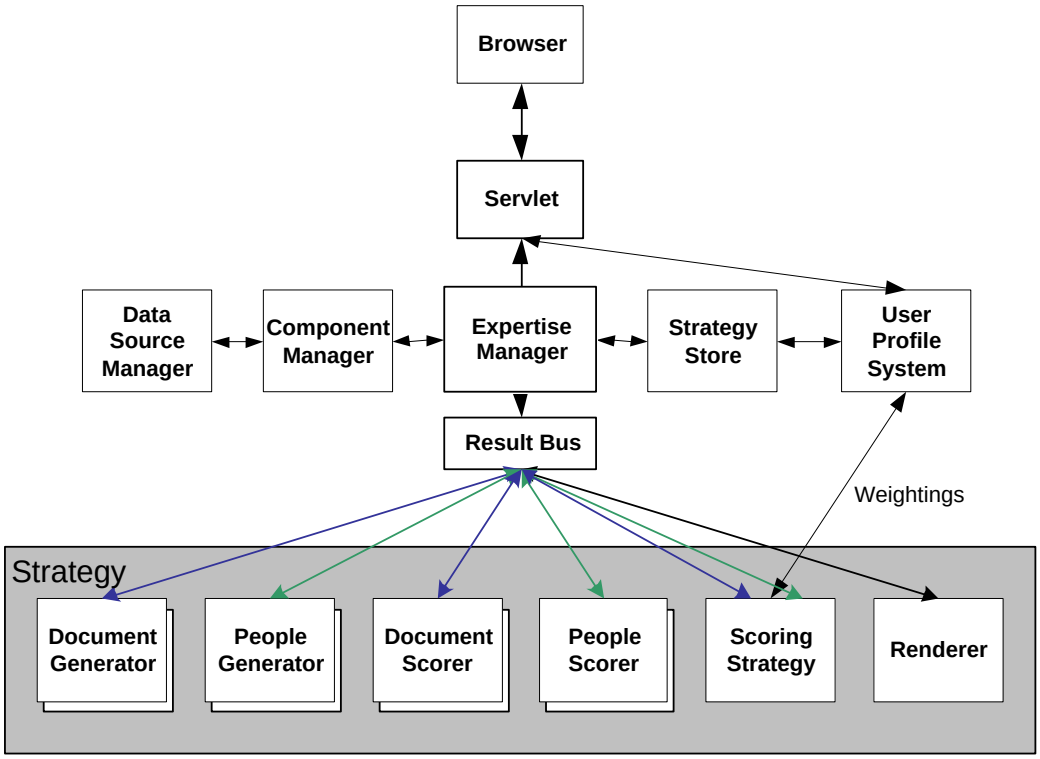


Figure 3. Expertise finder architecture

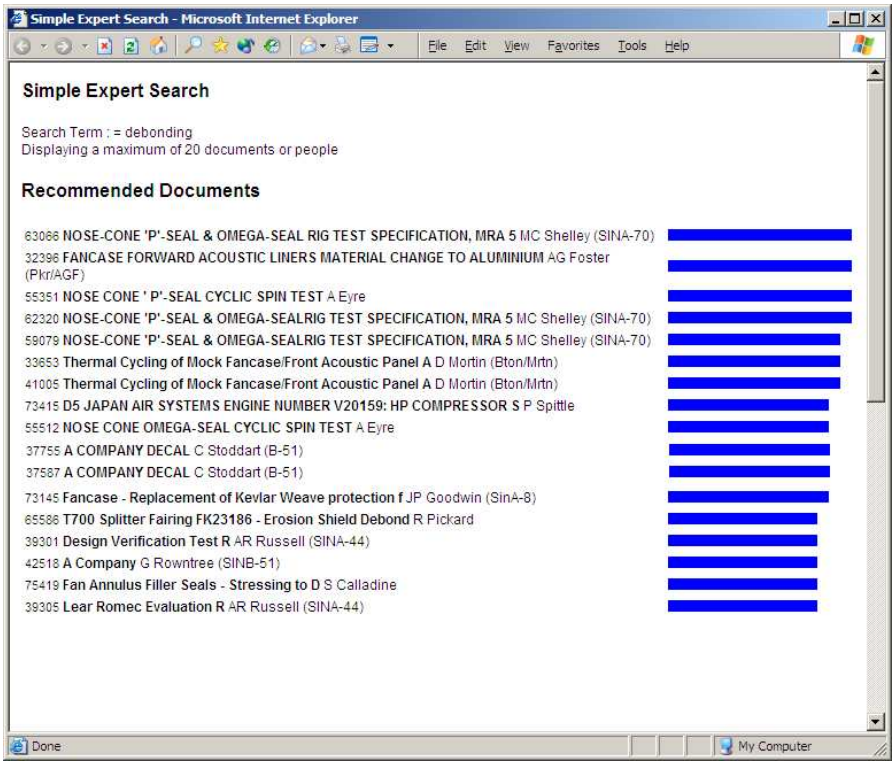


Figure 4. List of recommended documents, score is given by a bar graphic.

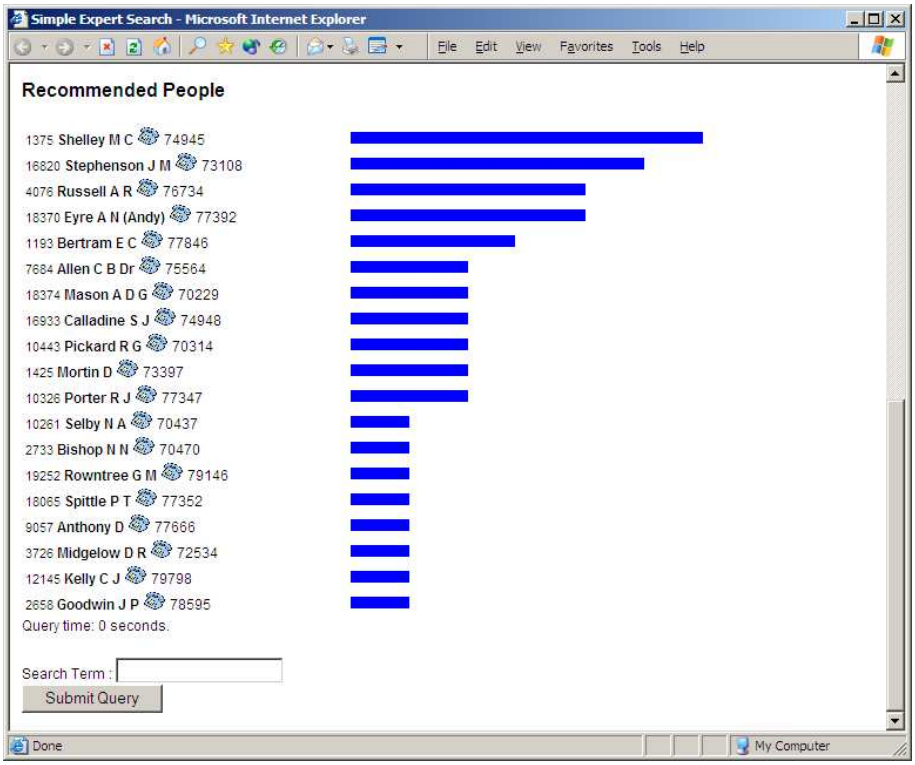


Figure 5. List of people with contact information, score is given by a bar graphic.

final score is an accumulation of all of these scores according to an interchangeable strategy that can be influenced by the user. Therefore the final score is relative and not comparable across strategies or users.

The Servlet component is responsible for generating the finished pages and providing the necessary responses to the Web browser. It passes the query to the Expertise Manager. This acts as the central hub and is responsible for starting the various components and running the requisite search *strategy*.

In the architecture the *generators* produce new results, either documents or people. They typically do this by interfacing with the existing systems within an organisation such as databases or document repositories. *Generators* will produce lists of results from a query and may provide a score for each result. For instance a *generator* based on a search engine might return a list of documents and the relevancy score given by the search engine. In this system all scores are a number between zero and ten. The final scores can be any value, this is in keeping with the nature of the system that the final scores are only relative. *Scorers* take results produced by *generators* and give a new score for each result depending on some other algorithm. For instance the same list of documents found by the full text search engine could be given a score based on their age. New documents might be given a higher score compared to older documents. This design implies an implementation relationship between *scorers* and *generators* and that each component is not totally independent.

The Expertise Manager runs a particular *strategy* to solve the query. The *strategy* manages the execution of the Components in a certain order and manages the manipulation of the data they return. For instance a *scorer* might need certain information about certain documents or can only function on a subset of results. Again the *strategy* component will be highly specific to the application and the organisation within which it is run. The EF Manager chooses which *strategy* to run at query time.

For every result found the system keeps all scores for that particular entry as well recording the origin of that score. For each document or person produced a *result* object is created with the following form.

$$Result = Source[id, origin] + n \times (Score[value, origin])$$

In each case the origin refers to the name of the Component as known by the Component Manager. The *results* list is passed to each Component which may make use of any parts of the data and add its own entries or scores as it sees fit.

The final list of *results* is given to the *ScoreCompiler* which is responsible for producing the final score. To produce this final score a list of ratings or multiplication factors are required. The ratings is a list of values which are used as multiplication factors for each type of score. The *ScoreCompiler* performs the simple calculation of multiplying each score by its factor then summing the results. It adds this to each *result* as a new *score* entry with the origin name Final. The *strategy* provides the ratings values but can also obtain or alter them by using the Preferences system and hence allow for individual user intervention in the final result.

The Preferences system is a general purpose way of storing preferences for each user. A simple API maintains a unique id for each user, managed using cookies, and allows the store of any key-value pair in a database against this id. Hence any part of the system can store and retrieve arbitrary data for an individual user. This can be used to override default values for the ratings given by the *strategy*.

The finished *results* are ranked and then used to produce the final display. This service is provided by a Renderer component. The current component produces HTML but other Renderers could provide other types of data such as XML.

The Data Source Manager controls access to underlying database connections in order to help cope with large scale deployments of the system.

Example application and implementation details

The current system is built around data supplied by a large manufacturing company. The major source comes in the form of databases of internal publications. These databases list details on internal technical reports produced at individual company sites. The fields in these databases are similar but not exactly the same. We also have a source of phone numbers and locations for all the employees of the company, this database has over 20000 entries. Neither data source can be guaranteed to be correctly maintained nor can the entries in any fields be guaranteed to be 'correct'. For instance people can edit their own entries in the phone book to give themselves any title or enter any name for their particular work group. Both are stored correctly elsewhere in other sources but not currently available.

The system we have developed comprises the following components, two document generators and a number of scorers. Each technical report database has been placed into MySQL and a Generator written to perform full text searches of these resources. This has required careful design as there are a total of 300,000 entries in the two databases. Unless care is taken the time to query such resources can be considerable and the number of results returned can easily overwhelm the rest of the system. We currently use the full text search capabilities of MySQL and use the relevancy scores it produces.

We use the phone book database as a People Generator. Again the raw data we received was placed into a MySQL table. For each document found by the Document *generators* an attempt is made to match the author against the phone book. This entails understanding the correct fields which hold the author data, that the formatting of the fields do not match and that the author field of the document databases are free text and highly inconsistent. This is a prime example of where the *strategy* code

is required to perform raw manipulation of the data in order to perform this matching process.

If the author is matched and the department they have given in the report database matches that in their phone book entry then the People Generator deems to have a strong match for that person, because they have not moved department since the report was written, and returns a high score. If the person is matched but not in the same department then the score is less. If there is no match then no Score is returned. This crude algorithm can be improved but data sources are required to aid in tracing and matching people. As such resources become available this Generator could be improved or replaced without altering the rest of the system. At such a time the scoring algorithm may need rebalancing to take account of the improvement of the accuracy of this Component.

We have developed a number of demonstrator Scoring components. One examines the date of the documents and gives them a score based on their age. Recent documents are given higher scores. The databases contain data for approximately twenty years of work so this is an important way to re-rank the data. Again there have been issues with attempting to work with free text fields and trying to understand the format of the document dates.

The new EF framework has been designed with the specific aim of allowing search strategies and ranking algorithms to be interchanged. This allows the system to be heavily customised to the particular social system found within an organisation. This is achieved by breaking down the process into steps performed by interchangeable components.

During the building of this new system it was realised that there cannot be general purpose ontologies through which all components communicate. There needs to be highly application specific code in the system to deal with the low level interchange and manipulation of data from the various sources in the system. The *strategy* code is the way of dealing with this problem. The code in this component is totally specific to a certain application of the framework and manipulates raw data from sources. It is therefore reliant on certain sources being available to work.

It is unavoidable that the *strategy* code is interlinked with the available components found at a particular site. It is not possible to design a single system that can be installed at a site with no additional work. Components need to be written to match available sources and the strategy needs to be written to match the social situation.

Instead of data exchange via ontologies the new system has a simple way to represent data. The data produced by one Generator or *scorer* may or may not be useful or recognisable to another. Therefore the data is represented simply by a unique id and the source of that id. If another Component wishes to make use of that data it needs to obtain it from the original component. This produces yet more interdependencies in the system design but clear API's to each Component help here. For instance a Scoring component might rank documents based on their date. Therefore the Generator that supplied those documents needs to be able to supply dates of documents via its API.

By dealing with the absolute minimum of data the system can handle relatively large numbers of results. In the previous system large amounts of data was generated as ontologies were instantiated and exchanged. A great deal of this data was often redundant to the final requirements. The decision to provide minimum results but providing full provenance is a trade off between results that are interchangeable against results that allow for speed and low overheads. The default usage of the system is to only provide a limited number of results, for instance the top twenty documents and people, therefore it is much faster to only extract the required data for those forty results at the end of the run.

Discussion

This paper reports the development of two expertise finders, the initial version based on Agents and the subsequent using a more simplistic architecture based around the need to heavily adapt the search strategy to the organisation. While the current system produces results of limited value, it demonstrates the infrastructure in use and successfully allows the *strategy* to utilise various document *generators* and *scorers* before computing a score. We are currently working on producing more meaningful *scoring* components.

The notion of trust in the results has been a key point made by our industrial partners at every step of the design process. For instance, in our first EF implementation the agents tried to match author names in a publications database against the live staff database. Those names that failed to match were returned to the user along with the names of those which did match. This crude example showed users exactly what was happening within the system and taught us a great number of implementation lessons.

The current system attempts to solve this problem with its transparency. Developers and users can see what scores are being given to each result and can see how these scores are brought together. All results are kept along with their origin for precisely this reason. It allows users to evaluate for themselves the whole process and to go into more detail on what one particular component has produced. This allows for possibilities such as saving the results of a query for caching and recalculating.

The process of finalising the details of a particular strategy will happen in two phases. There will need to be an initial requirements capture phase in which the users, or specific users with experience, are asked to explain the sources and methods that they would use to find an expert. From this a strategy component can be built which reflects a best attempt to turn this experience into algorithms and scoring schemes. We will then require an evaluation loop to refine this strategy.

One tool to help this process would be to be able to visualise

all of the score sets for each result. For instance a web page output could list all of the scores for each result as well as listing the ratings given to each contributing score. This will allow users to look deeper into the results, spot anomalies and have a much greater trust in the final results.

As expected our experiences have clearly shown that to produce an EF to support designers within a manufacturing organisation is a non-trivial undertaking. The main problem is the sheer scale and complexity of the organisation we have been worked with. Initially it took considerable amount of time before we felt we understood our partner and their actual processes. The second is the age and quality of the data we have been able to access. In this particular organisation knowledge is kept for many years, in many forms. Each area of the company uses and produces different types of information so no single solution will be appropriate organisation-wide, a major factor in the evolution of our design.

In conclusion our work has shown that it is possible to develop EF that rely on information currently available to organisations. With our approach companies do not need to undertake knowledge audit across the organisation to 'frontload'the system. In practice the viability of our approach is dependent on the ability to access all the information available in electronic format. With many organisations this can be rather spasmodic as the IT infrastructure has evolved over time with a range of conflicting priorities. Even with these caveats we believe that the introduction of this approach is feasible, though the challenges may be more of social than technical nature.

Acknowledgements

This work was funded in part by the University Technology Partnership for Design, which is a collaboration between Rolls-Royce, BAE SYSTEMS and the Universities of Cambridge, Sheffield and Southampton. The UK Engineering and Physical Sciences Research Council funded the work under grant number GR/M83582. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing official policies or endorsements, either expressed or implied, of the EPSRC or any other member of the UTP.

REFERENCES

[1] Y. Goa, I. Zeid, and T. Bardez. Characteristics of an effective design plan to support re-use in case-based mechanical design. *Knowledge based systems*, 10:337–50, 1998.

[2] D Khadilkar and L Stauffer. An experimental evaluation of design information reuse during conceptual design. *Journal of Engineering Design*, 7(4):331–9, 1996.

[3] R. Crowder, G. Wills, I. Heath, and W. Hall. Hypermedia information management: A new paradigm. In *3rd International Conference on Managing Innovation in Manufacture*, pages 329–34, University of Nottingham, 1998.

[4] E Auriol, R M Crowder, R J McKendrick, R Rowe, and T Knudsen. Integrating case-based reasoning and hypermedia documentation: An application for the diagnosis of a welding robot at Odense steel shipyard. In *Proceeding of the International Conference on Case-Based Reasoning (ICCBR'99)*. 1999.

[5] D. McDonald and M. Ackerman. Just talk to me: a field study of expertise location. In *Proceedings of ACM 1998 Conference on Computer Supported Cooperative Work, CSCW 98*, pages 315–24. ACM, New York, 1998.

[6] I Becerra-Fernandez. The role of artificial intelligence technologies in the implementation of people-finder knowledge management systems. In *Proceedings of the 2000 American Association for Artificial Intelligence Spring Workshop*. AAAI, March 2000.

[7] K. Bollacker, S. Lawrence, and C. Giles. Citeseer: an autonomous web agent for automatic retrieval and identification of interesting publications. In K. Sycara and M. Wooldridge, editors, *Proceedings of the Second International Conference on Autonomous Agents*, pages 116–23. ACM, 1998.

[8] P Chandrasekaran and A Joshi. An expertise recommender using web mining. In *Proceedings of AAAI 14th Annual International Florida Artificial Intelligence Research Symposium*. American Association for Artificial Intelligence, 2001.

[9] M. Ackerman and D. McDonald. Answer garden 2: Merging organisational memory with collaberative help. In *Proceedings of ACM 1998 Conference on Computer Supported Cooperative Work, CSCW 96*, pages 97–105. ACM, New York, 1996.

[10] D. McDonald and M. Ackerman. Expertise recommender: a flexible recommendation system and architecture. In *ACM 2000 Conference on Computer Supported Cooperative Work*, pages 231–40. ACM, New York, December 2000.

[11] R Crowder, G Hughes, and W Hall. An agent based approach to finding expertise. In D Karagiannis and U Reimer, editors, *Proceedings Fourth International Conference on Practical Aspects of Knowledge Management*, pages 179–88. Vienna, Austria, December 2002.

[12] L Moreau, N Gibbins, D DeRoure, S El-Beltagy, W Hall, G Hughes, D Joyce, S Kim, D Michaelides, D Millard, S Reich, R Tansley, and M Weal. SoFAR with DIM agents: An agent framework for distributed information management. In *Proceedings of PAAM00*. Springer-Verlag, 2000.

[13] T Gruber. Toward principles for the design of ontologies used for knowledge sharing. Technical Report KSL-93-04, Knowledge Systems Laboratory, Stanford University, August 1993.

[14] M. Weal, G. Hughes, D. Millard, and L. Moreau. Open hypermedia as a navigational interface to

ontological information. In H. Davis, Y Douglas, and D Durand, editors, *Proceedings of Hypertext'01*, pages 227–36. ACM, 2001. On-line at http://www.bib.ecs.soton.ac.uk/data/5161/pdf/ht01.pdf.