Intelligent Caching: Selecting, Representing, and Reusing Data in an Information Server*

Yigal Arens[1] and Craig A. Knoblock[1,2]
[1]Information Sciences Institute and
[2]Department of Computer Science
University of Southern California
4676 Admiralty Way
Marina del Rey, CA
310-822-1511
{arens, knoblock}@isi.edu

Abstract

Accessing information sources to retrieve data requested by a user can be expensive, especially when dealing with distributed information sources. One way to reduce this cost is to cache the results of queries, or related classes of data. This paper presents an approach to caching and addresses the issues of which information to cache, how to describe what has been cached, and how to use the cached information to answer future queries. We consider these issues in the context of the SIMS information server, which is a system for retrieving information from multiple heterogeneous and distributed information sources. The design of this information server is ideal for representing and reusing cached information since each class of cached information is simply viewed as another information source that is available for answering future queries.

1 Introduction

The SIMS project [1, 4] is addressing the problem of accessing information stored in heterogeneous distributed sources – databases and knowledge bases – in a convenient, transparent, and efficient manner. SIMS currently reduces the amount of information that must be retrieved by the use of syntactic and semantic optimization on the queries and the retrieval plans. In almost all cases some retrieval is still necessary, since at least the responses to the user's query may need to be transferred from remote information sources to the user's machine. It may be possible to avoid multiple retrievals of the same or related data from the same information source. This is particularly important, since transferring data over the network (typically the Internet) is often the most expensive aspect of the retrieval process.

To reduce the amount of data needed for retrievals we are working on an efficient and flexible approach to identifying and caching information that is likely to be usable in future queries. The idea is to store data related to that requested by a given query with the expectation that it will be relevant to future queries. This appears to be particularly promising, since SIMS is envisioned as being used as a tool in the execution of tasks: it is likely that a series of related queries will be issued in succession, with the results being at least partially overlapping, and sometimes even subsets of those of earlier queries. This is a common situation with cooperative query-answering systems [9].

There are several problems that must be considered in order to assure that data can be efficiently cached and reused:

1. First, one needs a scheme for selecting an appropriate set of information to cache based on a user's query. The standard approach is simply to cache either the exact data retrieved in response to a user's query or a subset of that data. We present an alternative approach that caches data that will potentially be useful for a much wider range of related queries.
2. Second, one needs a scheme to represent the cached information. When a future query is issued, it must be clear which information is already stored locally and which must still be retrieved externally. Since SIMS already uses a knowledge representation language to represent the contents of the information sources, it is quite natural to represent the contents of a class of cached information.
3. Third, one needs a mechanism for deciding when it is appropriate to reuse the cached information to answer a given query. The process of determining the relationship between what has been previously cached and what is needed must actually result in improved efficiency. This is handled in SIMS by treating each class of cached information as another information source. Then the underlying query processing facilities use the cached information when it is appropriate.

After providing an example of how SIMS processes a user's query, we will address each of these issues in turn. We present a set of principles for deciding which information should be cached and describe how the representation and use of cached information fits naturally into the SIMS framework.

*The research reported here was supported by Rome Laboratory of the Air Force Systems Command and the Advanced Research Projects Agency under Contract No. F30602-91-C-0081. Views and conclusions contained in this report are the authors' and should not be interpreted as representing the official opinion or policy of ARPA, RL, the U.S. Government, or any person or agency connected with them.

2 An Overview of SIMS

Given a query, SIMS identifies an appropriate set of information sources, formulates and executes the appropriate queries, and processes and presents the results to the user. Queries to SIMS are expressed in the Loom language. Loom [6, 7] is a member of the KL-ONE family of knowledge representation systems. In Loom objects in the world are grouped into classes and there are a set of relations that describe the relationships and properties of classes. See Figure 1 for a small fragment of the Transportation Planning domain model, which currently contains over 800 concepts. Classes are indicated with circles, relations with thin arrows, and subclass relationships with thick solid lines. Relations are inherited down to subclasses. Shaded classes are contained in some information source and correspond to the model classes to which they are linked by thick patterned lines. All classes can be used in queries, but they must eventually be translated into information-source terms (i.e., into combinations of references only to classes linked directly to shaded classes).

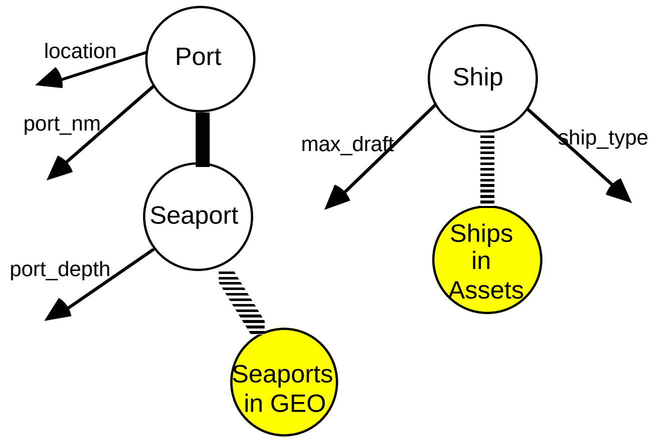

Figure 1: Model Fragments

Each class in the model define a set of "instances", which are fully described, specific members of the class. These instances are typically stored in one or more information sources and are retrieved to answer specific queries about a class. The language used to describe these classes (i.e., Loom) is also used as the SIMS query language. By that we mean that a query to SIMS is in the form of a definition of a (typically new) class of objects in the domain. As a simple example, the user may issue the query shown in Figure 2. This query requests names of all ports in Britain whose depth is sufficient to accommodate all ships of type "Transport". The query uses terms from the Domain Model (i.e., not those relating to information sources specifically) to essentially describe a new class – those ports in Britain that can accommodate the ships of interest.

Currently, SIMS processes such a query following the steps listed below:

1. Identify Seaport and Ship as the classes of objects relevant for this query.
2. Find classes contained in some information sources that correspond to them.
3. Check if the relations on the classes referenced in the query are in fact described in the candidate information sources as well. (We assume that is the case here).

```
(retrieve (?port_name)
        (:and (seaport ?port)
              (location ?port ``Britain'')
              (port_nm ?port ?port_name)
              (port_depth ?port ?depth)
              (ship ?ship)
              (ship_type ?ship ``Transport'')
              (max_draft ?ship ?draft)
              (< ?draft ?depth)))
```

Figure 2: Example Query

4. Determine if and when joins must be performed and that the available information supports them (e.g., in this case, that the units of depth and draft are the same).
5. Send appropriate queries to the respective information sources (the LIM/IDI [8, 10] system is used to generate queries to relational databases) and manipulate the results as necessary to obtain the precise data requested by the user.

Generally speaking, SIMS attempts to obtain the required information using as few information sources as possible and to avoid joins performed locally (in Loom). Joins within Loom are more costly – both because data from several sources must be moved to Loom, and because Loom's reasoning is slower. Details, including discussions of what happens if the simplifying assumptions made above are not met, are provided in [1].

2.1 What to Cache in this Example?

Offhand, one might consider the classes listed below as potential ones to cache for future use. SIMS' analysis of the model and of the user's query enable it to determine that they can all be retrieved by minimal modifications to the queries that in any case need to be executed to satisfy the user's request.

Class 1. All seaports.

Class 2. Seaports in Britain.

Class 2a. Seaports in some larger geographic area including Britain.

Class 3. Seaports in Britain with a depth greater than the maximum draft of any ship.

Class 4. Seaports in Britain (or in a larger area) with a depth greater than the maximum draft of any transport ship. (This is what the user's query requested.)

Class 5. All ships.

Class 6. Transport ships.

Class 6a. A larger subclass of ships.

Class 7. Transport ships with a draft less than the minimum depth of any seaport.

Class 8. Transport ships with a draft less than the minimum depth of any seaport in Britain.

In the sections that follow, we discuss the problems of caching. In the course of doing so, we will refer to this example and provide a basis for deciding which (if any) of the above classes should actually be cached.

3 Representing Cached Information

Having retrieved the information requested in the user's query, it might be useful to cache it for future reuse. But cached data is useless unless the information server can determine precisely what is cached locally and how it is related to the data that must be retrieved in response to a new query. This means both that a principled method for representing the contents of retrieved collections of data must be used, and that the reasoning ability necessary to support the identification of relevant, existing cached information is supported.

A knowledge representation system can very naturally represent and reason about classes of cached information. In SIMS this is particularly appropriate since we already use the LOOM knowledge representation system for querying and modeling purposes. When it is eventually decided what collection of information to cache, the corresponding concept class is defined in Loom and the data are stored as instances of that class. This new class of cached information becomes another information source for the system to use in future queries. An example of this will be shown later.

We want to avoid caching information that cannot be easily described in the semantic model. Recall that our example query involved a join. This join does not exist only at the "information source level" (i.e., when viewing the query in terms of the information sources that will ultimately be used to answer it). The join is expressed at the domain level as well: in the query's high level form itself. We try to avoid caching more complex classes like those that involve such joins for two reasons. First, when the class of objects a query defines is overly complex, the chance that it will substantially overlap with that defined by a future query is reduced. The expected gain from storing such classes of instances would thus be small. Second, beyond the mere question of representation, we must consider also reasoning with the resulting classes. Whenever a new query is posed, all of the cached classes of data that already contain needed items must be identified, and exactly which additional data still needs to be retrieved from remote sources must be deduced. When the definitions of the cached classes are overly complex, such reasoning becomes elaborate and very time consuming.

In some cases a simple domain-level concept corresponds to information from a variety of information sources. This would be an useful class to cache since the cost of combining this information could be expensive, but the class is simple to describe and reason about and might be used frequently.

Complexity considerations cause us to rule out Classes 4 and 8 from Section 2.1 as caching candidates. Classes 3 and 7 are slightly different cases. Although they appear to involve joins, they do not, since SIMS' learning mechanism produces abstracted information about information sources, such as the maximum depth of ships in it. Classes 3 and 7 therefore are just further restrictions on the classes of Seaports and Ships, respectively. They remain caching candidates.

4 Selecting the Information to Cache: Principles of Intelligent Caching

Having decided how to represent retrieved data still leaves us with the problem of deciding which data to retrieve and cache. It might appear at first glance that one should simply consider caching the data for which the user's query asked, or some structurally/syntactically determined subset of it – after all, it is going to be retrieved anyway. This approach is taken in other systems [3, 11, 2]. It is the only reasonable approach available when the only way to index/classify the cached data is by the query it corresponds to. However, it raises two problems mentioned earlier: such data is unlikely to be useful except where the identical query is issued repeatedly, and comparing it to new queries may be difficult. Given the rich semantic knowledge available to our system, we can relax and restrict queries in a variety of ways while remaining close to the user's original.

Let us call the class of instances requested by the query Q, and the class to be cached C. Ultimately, some balance of the following principles must hold:

Principle 1: C should not be too large in size. While databases are designed to deal with very large quantities of data, the knowledge representation system and its reasoning facilities typically are not.

Principle 2: C's definition in the knowledge representation language should be relatively simple. The more complex the definition, the less likely it will be useful for future queries and the more expensive it will be to reason about.

Principle 3: The difference between C and Q should be relatively small. One wishes to minimize the amount of extra data that must be transmitted and stored, so C should not contain much more than Q. At the same time, C should not contain much less than Q, so as to maximize the future benefit of the caching.

Principle 4: C should be retrievable without recourse to other information sources than those required to obtain Q. Again, one wishes to minimize the amount of extra work performed, beyond that required to answer the user's query.

Needless to say, the data cached will be biased by the queries seen by the system. This flows from the obvious rationale behind caching: that information already asked about is more likely to be of interest again. Our approach to the design of an information server enables us to keep track of which cached classes are used again and how often. Unneeded ones can be purged.[1]

We are now in the process of implementing the caching scheme described in this paper. In accordance with the principles listed above, we are implementing the following approach in selecting the classes of data to be cached:

- Before processing queries, we define a threshold for the size of a cached class. Any class that exceeds this threshold will not be cached. The data may have to be retrieved if it is required to satisfy the user's query, but it will still not be cached. As noted earlier, the knowledge representation system is not designed to efficiently handle very large amounts of data.

  In addition, we define a threshold for the relevance of the information in a class. Each class consists of a set of relations. Each of those relations may or may not be relevant to any given query. We can then keep track of how often each relation on a class is queried. In the example above the port-name, depth, and location are relevant to the port class. The threshold is then used to decide whether the relations on some class have been

[1] A similar reevaluation and purging process takes place with SIMS' learned reformulation rules [4].

accessed frequently enough in previous queries to be considered relevant. If certain information has never been accessed, then it is probably not worth retrieving and caching.

- For a given query, the server splits the query into components, each corresponding to the individual domain-level concepts that comprise the query. This is a simple way of ensuring that retrieving the data desirable for caching will not involve any of the more costly operations, such as joins between tables in separate information sources.
- Increase the utility of cached data by retrieving more information than was requested for a given class. This is done by retrieving other information in the class that is above the relevance threshold defined above. Of course, additional information is only retrieved if it does not exceed the overall size threshold.
- Next, the server simplifies the structure of the retrieved class by relaxing the query by eliminating one or more constraints. This typically means retrieving more data. The choice of which constraints to drop is done so as to get as much information as possible without violating the first principle in this list.
- Then the server increases the potential utility of cached data by broadening constraints on the query. For example, if a query asks for some features of all ports in Britain, we consider retrieving those features for all ports in the United Kingdom, or in Western Europe. Consider a more sophisticated example: given a user query that asks for all ports in Britain that are sufficiently deep to accommodate any ship of a certain type. In this case a different class could be cached – that of all British ports with a channel depth greater than X, where X is the maximal draft of ships of the relevant class. The new class, even though equivalent to the old one, may be more useful by virtue of its simpler definition. SIMS' rich semantic representation of the domain and information sources allows it both to come up with potential broadenings of query constraints, and to easily determine whether the additional data retrieved will still come from the same information source. The choice of which constraints to broaden is, again, done in a way that does not violate the first principle.

Following these strategies, Classes 1 and 5 are eliminated as caching candidates since the number of instances in them is large. Classes of the type 2, 2a, 6, 6a and 7 will be eliminated if their size is too large or they cannot be retrieved from the same information source that is already being accessed. Since the Seaports table in GEO contains all ports in Western Europe(see Figure 1), we will choose to cache all such ports – assuming the number is below our threshold. Under such circumstances, there would be no need to consider Class 3, since it is smaller than the one chosen for caching.

Our new model will look as shown in Figure 3. A new information source class will be created, Local W. Europe Seaport, containing the cached instances. The precise characterization of the contents of this new class will be encoded by creating a new domain class, W. Europe Seaport, to which the cached class is linked. West European seaports are distinguished from seaports in general by having their location property filled appropriately (the model figure does not include an indication of this). The newly cached class is shaded, to indicate that it too will be treated as an information resource in the future. More on that subject next.

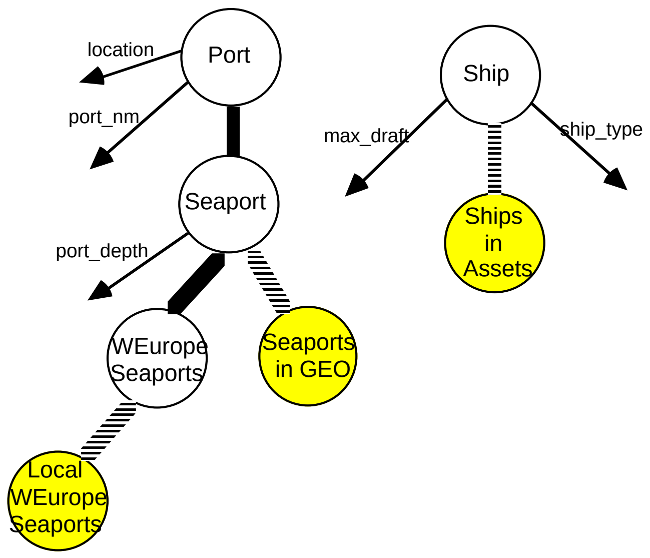

Figure 3: Extended Model

5 Reusing Cached Information

SIMS views data as being located in information sources. These include databases and knowledge bases. When data is cached, the new concept created to represent the class of in question is marked as a new information source. When processing a new query and finding sources of information relevant to it and an access plan for querying them, SIMS' planner simply treats each cached class the same as it does any information source.

When planning for a query, SIMS takes account of the costs of accessing each information source. For a typical database, the costs involved are estimated from the anticipated size of the retrieved data and its transmission time. In the case of a cached class, the costs are based on the size of the class and the processing time that would have to be devoted by the knowledge representation system used. The system naturally gravitates towards using the cached information where possible. But on balance, SIMS may find sometimes that it is more efficient not to use cached data. This may happen, for example, if a query to the database from which the data was cached would be necessary in any case: it then may be best to simply augment that query to get the cached data rather than spend the knowledge representation reasoning time needed to filter it out of the locally stored cache. However, in those cases where the cached information eliminates the need to query an information source, the use of the stored information has the potential to provide significant efficiency gains.

If, for example, SIMS is now given the query shown in Figure 4, it will go through the same planning process described earlier in Section 2. However, when it attempts to determine which information sources contain French ports, there will be two candidates – unlike in the earlier case. The two sources will be Seaports in GEO and W. Europe Seaports (Figure 3). It will be cheaper to get the data from the locally cached source, since all data needed for the query is already in it. Access to one information source will thus

```
(retrieve (?port_name)
        (:and (seaport ?port)
                (location ?port ``France'')
                (port_nm ?port ?port_name)
                (port_depth ?port ?depth)
                (ship ?ship)
                (ship_type ?ship ``Transport'')
                (max_draft ?ship ?draft)
                (< ?draft ?depth)))
```

Figure 4: Example Query

be completely eliminated from the original query plan. It is interesting to note that the data returned for this query is not even a subset of obtained for the first one – the typical situation caching is envisioned for.

6 Related Work

The most closely related work on caching in the database literature is the work by Finkelstein on the topic of common subexpression analysis [3]. Finkelstein presents an approach to reusing previously retrieved data to answer queries or subexpressions of queries. His approach is to cache the results of queries and then look for subexpressions of a new query that match classes of previously cached data and reuse that data where appropriate. The absence of semantic modeling makes it difficult to represent the cached information and does not permit Finkelstein to consider the kind of perturbations to the classes requested in the query that we consider here. He can cache only the complete results from some query and hope that it will be useful to evaluate a subexpression of some future query. It is not clear from the description given in the paper how or where the cached results (Temporaries) are stored and if there is an effort to manage their size and limit the length of time they are maintained.

There has been other work that deals with the issue of caching in an information server environment, but this work has focused on the issue of maintaining the consistency of client cached data [11, 2]. This is an important issue, but it is orthogonal to the issues in this paper of what to cache, how to represent it, and how to reuse it.

7 Discussion

This paper presents our ideas concerning principles that underlie caching of information retrieved when executing user queries in the context of a knowledge-based information server. Some of the issues and approaches raised in the paper are of general concern and applicability in any system that attempts caching. Others are more specific to SIMS-like servers that have at their core a rich semantic representation of the application domain and the available data resources. However, it is our position that such a semantic model is an absolute requirement for any serious attempt to attack the problem of accessing heterogeneous distributed information sources in a convenient, transparent and efficient manner. We demonstrate the capabilities of such modeling in [1].

We believe that the issue of caching will become even more pressing in the near future. Computer networks are spreading at a high rate, and their use for disseminating information is growing. Networks like the Internet are very slow when compared to communication directly with a local DB. Furthermore, the relatively primitive database management and querying facilities available (in systems such as Gopher and WAIS), often result in much larger data sets being transmitted than are actually requested. A keyword-based query for books by "John McCarthy" will also retrieve books about John McCarthy. Currently, we compensate for this in SIMS by analyzing the retrieved data, classifying it in our knowledge representation system, and then filtering out only those instances that precisely match the original query. In such an environment, caching can provide an even greater advantage – saving the transmission of far more data than a query itself might suggest. If the Internet does get used for series of related queries, the benefits will be substantial.

One important point that has not been extensively discussed here is how long cached information should be maintained. There are two considerations involved: First, we wish to avoid caching too much information in the knowledge representation system, as mentioned earlier. Second, databases get updated, and the cached information will likely not remain valid indefinitely [11, 2]. To address the first point, we intend to employ a system of tracking the actual usage of cached classes and eliminating those that are not used often enough. This follows a process similar to that which we already use with learned abstractions of information source contents, structures that are used elsewhere in SIMS [5]. The second point cannot be addressed until SIMS includes in its Loom representation of information sources details about their update frequency and the temporal persistence of their contents. We view these topics as areas for future research.

References

[1] Yigal Arens, Chin Y. Chee, Chun-Nan Hsu, and Craig A. Knoblock. Retrieving and integrating data from multiple information sources. *International Journal on Intelligent and Cooperative Information Systems*, 2(2):127–158, 1993.

[2] Michael J. Carey, Michael J. Franklin, Miron Livny, and Eugene J. Shekita. Data caching tradeoffs in client-server DBMS architectures. In James Clifford and Roger King, editors, *Proceedings of the 1991 ACM SIGMOD International Conference on Management of Data*, pages 357–366, Denver, CO, 1991.

[3] Sheldon Finkelstein. Common expression analysis in database applications. In Schkolnick, editor, *Proceedings of ACM SIGMOD*, pages 235–245, 1982.

[4] Chun-Nan Hsu and Craig A. Knoblock. Reformulating query plans for multidatabase systems. In *Proceedings of the Second International Conference of Information and Knowledge Management*, Washington, D.C., 1993. ACM.

[5] Chun-Nan Hsu and Craig A. Knoblock. Rule induction for semantic query optimization. In *Proceedings of the Eleventh International Conference on Machine Learning*, San Mateo, CA, 1994. Morgan Kaufmann.

[6] Robert MacGregor. A deductive pattern matcher. In *Proceedings of AAAI-88, The National Conference on Artificial Intelligence*, St. Paul, MN, 1988.

[7] Robert MacGregor. The evolving technology of classification-based knowledge representation systems. In John Sowa, editor, *Principles of Semantic Networks: Explorations in the Representation of Knowledge*. Morgan Kaufmann, 1990.

[8] Donald P. McKay, Timothy W. Finin, and Anthony O'Hare. The intelligent database interface: Integrating AI and database systems. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, Boston, MA, 1990.

[9] M.A. Merzbacher and W.W. Chu. Query-based semantic nearness for cooperative query answering. In *In Proceedings of the ISMM International Conference on Information and Knowledge Management, CIKM-92*, pages 361–368, Baltimore, MD, November 1992.

[10] Jon A. Pastor, Donald P. McKay, and Timothy W. Finin. View-concepts: Knowledge-based access to databases. In *Proceedings of the First International Conference on Information and Knowledge Management*, pages 84–91, Baltimore, MD, 1992.

[11] Kevin Wilkinson and Marie-Anne Neimat. Maintaining consistency of client-cached data. In Dennis McLeod, Ron Sacks-Davis, and Hans Schek, editors, *Proceedings of the 16th International Conference on Very Large Data Bases*, pages 122–133, Brisbane, Australia, 1990.