The HyperHotel Application Built over D.I.C.E. and Co.In.S.

Peter Triantafillou and Nikolaos Ntarmos
Computer Engineering & Informatics Dept.
University of Patras
<peter,ntarmos>@ceid.upatras.gr

John Yannakopoulos
Computer Science Dept.
University of Crete
<giannak@ics.forth.gr>

Abstract

Content integration of web data sources is becoming increasingly important for the next generation information systems. However, all proposed solutions are faced with the same performance bottleneck: the network overhead incurred when contacting the integrated e-sites. With this demo paper we shall demonstrate the functionality of HyperHotel. HyperHotel is used for finding appropriate hotel rooms when travelling. Its novetlies are that it is designed and implemented as an internet web-hotel content integration application and that it is built on top of D.I.C.E. and Co.In.S.; a novel content integration infrastructure consisting of a domain-independent COntent INtegration System and its Data Integration Cache Engine. We'll show how the infrastructure of D.I.C.E. and Co.In.S. can be applied and exploited in HyperHotel in order to improve the response time of complex user queries. This exemplifies the significance of this infrastructure since HyperHotel is representative of a large class of e-commerce, content integration applications.

1 Introduction

The unstructured and heterogeneous nature and the immense proportions of data currently available through the web, make information integration a vital part of many modern data management systems and data warehouses. The most wide-spread approach to automated data integration and dissemination utilizes a mediator and wrappers for the back-end sites. Relevant research usually focuses on the wrapping part of the integration process. In the real-world, however, the bottleneck of the system is the network overhead, incurred when contacting the integrated e-sites.

Co.In.S. – the Content Integration System, is a domain-independent mediator-based system, featuring D.I.C.E. – a caching engine based on novel techniques. DnC[1] targets content integration systems, representative of those needed to support a large class of e-commerce applications (e.g. comparative e-shopping for books or CDs, e-hotels, online auctions, etc.), with such characteristics as: (i) the need for translation of the large variety of different data models, used by the back-end e-sites, into a single data model, (ii) the fact that queries posed to the system are over all relations of the unified data model and involve all attributes in the joined relations, and (iii) the fact that the unified model and query semantics are available a-priori at the mediator.

The key characteristics of D.I.C.E. include: (i) the use of active caching ([2]) techniques in the context of content integration – a context much more complex than that of traditional web proxy caching – as well as the use of semantic information to achieve *fully semantic active caching* in the specified context, (ii) the use of materialized views over cached entries to extract the *maximally contained* cached query results, with the minimum possible complexity, and (iii) a novel use of Bloom filters ([1]) and of other related novel data structures in the hit/miss detection algorithm and in the production of queries (refered to as *subqueries*) corresponding to the exact missing content, greatly reducing the cost of these procedures.

2 D.I.C.E. & Co.In.S Architecture

The goal of the DnC system is to build a domain independent, dynamic mediator system with caching, with an emphasis on speed, platform independence, and ease of deployment. For modularity and ease of maintenance, DnC is built as a set of independent but cooperating components. A schematic view of DnC is presented in fig. 1.

Briefly, DnC is a multi-tier architecture, consisting of: (i) the user interface, (ii) the Front-end modules, implemented using Java Servlets and XML-related technologies, (iii) the caching subsystem (aka D.I.C.E.), based on off-the-self database management systems and novel cache management techniques, (iv) an XML-enabled mediator, utilizing third-party wrappers and standard XML-based technologies, and (v) the back-end e-sites. All internal commu-

[1] We use DnC to refer to D.I.C.E. and Co.In.S., where appropriate.

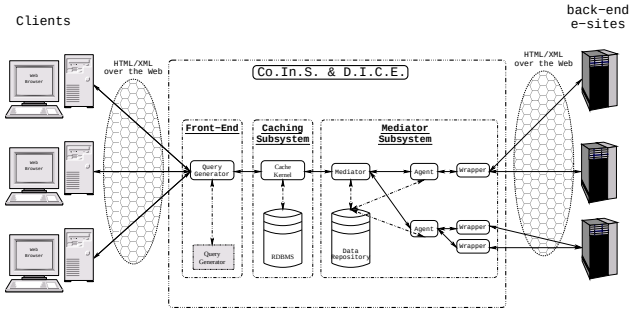

Figure 1. DnC Architecture

nications between the components of DnC use XML documents, constructed according to a set of predefined DTDs: the *Query* and *Response DTD* respectively[2].

2.1 D.I.C.E.: The Data Integration Cache Engine

Data integration systems solve the problems of heterogeneity of available content and of selectively querying multiple sources. Such systems usually feature sophisticated algorithms for data extraction and conversion and strive to support the latest standards in the fields of query languages and data representation. However, the real bottleneck in such systems is the network overhead.

An effective and transparent method for tackling this class of problems is to cache queries and their results within the data integration system in an intelligent and efficient way. D.I.C.E., a vital part of our system, deploys efficient and flexible active semantic caching mechanisms for our data integration infrastructure.

D.I.C.E. uses a RDBMS at its storage layer. Cached information is stored in tuples in the RDBMS's relation and cached queries are represented in the cache as materialized views over the database relations.

The core idea of D.I.C.E. is the use of summarization of cached entries, based on a novel utilization of Bloom filters, along with specialized data structures, and of materialized views, in order to swiftly identify full and partial cache hits of query results, and to compute the set of queries whose results represent the data items not present in the cache (called "missing queries"). Our approach allows for very fast hit/miss identification and missing query generation. We'd like to stress that our caching framework provides the mediator with the **exact** set of missing queries and delivers the **exact** query response to the front-end modules (full semantic active caching).

The characteristics of queries that can be handled by our cache are shown in fig. 3. We assume that the projection_list includes all attributes contained in the union of the tuples in each of the target_relations (i.e. all possible attributes).

SELECT *projection_list*
FROM *target_relations*
WHERE (NOT) *atomic_constraint (atomic_attribute, value)*
[AND | OR] (NOT) *other_atomic_constraints*
AND (NOT) *comparison_predicate_constraint*
[AND | OR] (NOT) *other_comparison_predicate_constraints*

Figure 3. Class of queries supported by D.I.C.E.

Cache consistency and temporal coherency are issues typically discussed in works focusing on caching of data gathered from web sources. These notions are crucial to data-centric applications, like the ones supported by our infrastructure. In this work, we have an orthogonal attitude toward these problems. The requirements of these two concepts are highly dependent on the application semantics. We believe that some form of invalidation mechanism (e.g. TTL-based a la web proxy caching) will fit well with the rest of our system. In any case, the adoption of push-pull techniques or of approaches based on expiration mechanisms for dealing with the cached data invalidation problem effectively, represents an opportunity for a possible future extension to our framework.

2.2 Request Processing

To illustrate the functionality of DnC, we will now describe the typical steps of processing of a user query.

1. The DnC server exports, through its front-end modules, an HTML-based query interface. The user contacts the server, through her favorite web browser and poses her query. The front-end modules convert the user request to a QDC-XML document describing the query, and forward it to the caching subsystem for further processing.
2. The caching subsystem checks its internal data structures for whether there is a cache hit. In the case of a full hit, the cached result is returned to the front-end modules. In all other cases (partial/full miss), the caching subsystem constructs a set of subqueries, representing the missing data, and forwards them to the mediator subsystem.
3. For each such subquery, the mediator subsystem uses its semantic metadata repository to select the set of back-end sites that have to be contacted in order to answer the given queries, and retrieves the relevant web pages. After the wrapping process, the translation of information to the system's internal data model, and a preliminary query processing on behalf of the mediator (e.g. duplicate elimination, conversion of semantically

[2]We'll use the terms *QDC-XML* document and *RDC-XML* document to refer to a *Query/Response DTD Compliant* XML document respectively.

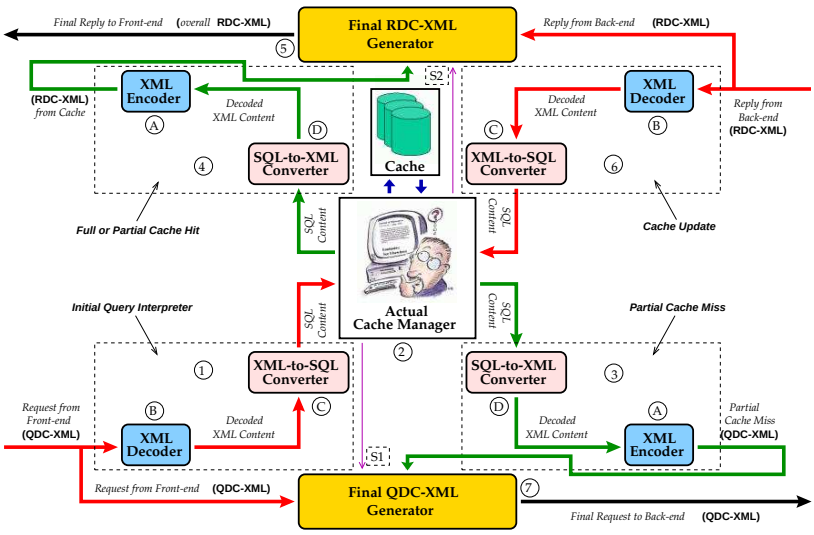

Figure 2. D.I.C.E.: The Cache Manager

identical tuples to a common format, etc.), the resulting RDC-XML documents are returned to the caching subsystem.

4. The caching subsystem stores the newly acquired data and updates its internal data structures. It then constructs a single RDC-XML document, combining all of the relevant cached tuples, and returns it to the front-end modules.

At this point, the request processing has ended. The front-end modules take on from here to pretty-format the output and to create appropriate HTML pages in answer to the initial user query.

3 Demonstration of the system

A classic example of data freely available through the world-wide-web in unstructured or semi-structured form, is data about accommodation facilities such as hotels and rent-rooms. The common practice for such companies is to have a web site offering location-related information (such as address, transportation etc.), room rates, availability checking, online booking etc. In order to demonstrate the functionality of DnC, we have built a web content integration application based on it – the HyperHotel.

3.1 HyperHotel: An example application

The HyperHotel is a data integration application, bringing together the vast amounts of information available online by real-world hotels. Users have the ability to search for their residence-of-choice in multiple e-hotels, without having to visit the web sites of each and every one of them.

Figure 4. The HyperHotel GUI

HyperHotel's goal is to answer to this demand in a semi-automated way, surpassing currently available paradigms utilizing data-entry and/or proprietary communication protocols between the integrator and the back-end sites.

For the moment, HyperHotel integrates web content from approximately 100 e-hotels, spread out in five major cities in Greece (namely Athens, Thessaloniki, Chania, Heraklion, Corfu). Users of HyperHotel pose their queries through a HTML form (fig. 4), based on such attributes as location, class, and facilities of the hotel, desired number of beds, desired price range, and desired date range. The result set is again presented in HTML format (a screenshot is omitted due to space considerations) and consists of tuples of hotel rooms matching the user constraints, along with a

link to the corresponding e-site.

To achieve better performance, HyperHotel prefetches, integrates, and stores all "static" data (i.e. data available through static HTML pages) at start-up, while still offering the capability of online dynamic querying through HTML form interfaces. User queries can include such attributes as hotel location, hotel class and facilities, number of beds per room, desired time period, and desired cost range. Constraints concerning time period and desired cost are range-based, while all other constraints are equality based (e.g. constraints on dates are always of the form "$date_i \{< \mid > \} value_i$", while constraints on hotel class are always of the form "$hotel\_class_i = value_i$").

3.2 Example Query Workload

We would like to stress here that HyperHotel is much more than yet-another price comparator; Because of D.I.C.E., the amount of data needed to be transfered for each query is cut down to a minimum. For example, let's assume that the cache is cold and a query Q_1 for 2-bed rooms in A or B class hotels located in Athens, for the period from August 1^{st} to August 15^{th}, with a maximum price of 100 Euros per day, is posed to the system. The query would be expressed using a QDC-XML document and sent to DICE for processing. The cache kernel would see that no relevant data is stored in its repository and would forward the query verbatim to the mediator. The latter would then contact the appropriate e-sites, integrate and filter the results and return them to DICE, which would update its repository and then forward them to the front-end for proper formatting.

Now assume that a second query Q_2 is posed to the system after Q_1, concerning 2-bed rooms in A class hotels in Athens, for the period from August 10^{th} to August 20^{th}, with a price ranging from 50 to 100 Euros per day. The result-set for Q_2 is obviously a subset of the result set of Q_1 for the time period until August 15^{th}. DICE would notice this and construct a set of subqueries concerning the data required to answer Q_2 but missing from the cache (i.e. 2-bed rooms in A-class hotels in Athens, for the period from August 16^{th} to August 20^{th} with a price range from 50 to 100 Euros per day). It's this set of subqueries that will then be sent to the mediator for integration and further processing.

Finally, assume that a third query Q_3, concerning 2-bed rooms in A-class hotels in Athens, from August 8^{th} to August 17^{th}, with a price ranging from 60 to 80 Euros per day, is entered in the system after Q_1 and Q_2. As can be easily seen, none of the result sets of Q_1 and Q_2 can answer Q_3 completely, but their union is a superset of Q_3's result set. DICE will use the cached data to answer Q_3 and will not engage the mediator for this query.

The demonstration will include query workloads such as this to exemplify the efficiency and the offered services.

4 Related Work

Currently there are several e-hotel price comparators available on-line (e.g. [4, 5, 7]). All of these sites are commercial and don't disclose information concerning the inner workings of their architectures. To the authors knowledge, it appears that these sites are using some kind of centralized repository (e.g. RDBMS) for the data they manage, probably maintained and updated after manual intervention, in cooperation with the back-end e-sites.

DnC and HyperHotel, as already explained, move several steps forward. First, in DnC data is gathered from the e-sites and integrated at query time when appropriate, without requiring some proprietary communication protocol or a-priori contract with the e-hotel. Second, DnC and HyperHotel feature novel caching techniques, using subqueries for partial matches; this results in tremendous efficiency gains when compared to bare-bone mediator-based systems, while keeping cached data fresh and up-to-date. Third, HyperHotel is relieved from the burden of maintaining the consistency of integrated data: this chore falls onto the owners of the integrated e-site (or their web host) to ensure their data's frreshness.

With respect to the research novelty of the infrastructure, we claim that it is the only to offer full active semantic caching, allowing the identification of the exact maximum contained partial cache hit and the missing queries to completely answer the user's query (as opposed to [6, 3]), in a setting representative of a large class of e-commerce applications.

5 Conclusions

In this paper we propose to demonstrate the D.I.C.E. and Co.In.S. system, using the HyperHotel application, built on top of DnC. DnC is a content integration system built to facilitate efficient web content integration, utilizing novel caching techniques for dynamic web content.

References

[1] B. Bloom. Space/time tradeoffs in hash coding with allowable errors. In *Communications of the ACM*, volume 7 of *13*, pages 422–426, July 1970.

[2] P. Cao, J. Zhang, and K. Beach. Active Cache: Caching Dynamic Contents on the Web. In *Proc. Middleware '98*.

[3] S. Dar, M. Franklin, B. Jonsson, D. Srivastava, and M. Tan. Semantic data caching and replacement. In *Proc. VLDB '96*.

[4] Hotwire.com. http://www.hotwire.com.

[5] Lodging.com. http://www.lodging.com.

[6] Q. Luo and J. Naughton. Form-Based Proxy Caching for Database-backed Web Sites. In *Proc. VLDB '01*.

[7] Travelocity. http://www.travelocity.com.