CASE-BASED REASONING AND LEGACY DATA REUSE FOR WEB-BASED RECOMMENDATION ARCHITECTURES

Francesco Ricci, Nader Mirzadeh, Adriano Venturini and Hannes Werthner*

eCommerce and Tourism Research Laboratory - ITC-irst
via Sommarive 18
38050 Povo, Italy
{ricci,mirzadeh,venturi,werthner}@itc.it

Abstract: *This paper describes a software framework for developing case-based reasoning (CBR) components that are seamlessly integrated into an enterprise architecture. The framework exploits recent standardization initiatives in the area of XML databases and mediator systems. We show how a legacy database or an XML data source, which may be tagged using a standard vocabulary, can be easily reused as case base with minimal additional efforts. An application of the proposed framework to the development of a case-based recommendation system for tourism travel planning is illustrated.*

1. Introduction

Case-Based Reasoning (CBR) is a problem solving methodology that takles a new problem by first retrieving a past, already solved similar case, and then reusing that case for solving the current problem. In a CBR recommendation system [5,10] a set of suggested (recommended) items is retrieved from the case base by searching for recommendation session cases similar to a target situation described by the user interacting with the system.

CBR is largely based on the notion and implementation of case similarity and similarity based retrieval from a case base. Case bases have been traditionally implemented using ad-hoc data structures loaded in memory at system start up. However, integration of CBR with RDBMS is becoming more and more important for many reasons, e.g., scalability; enterprise application

* This project is partially funded by CARITRO foundation.

integration; reuse of previous data. An initial set of works have therefore addressed this issue [17,5,10,15].

Notwithstanding these successful examples, and many others, there is still one major obstacle to the widespread application of such technologies. The quoted efforts to integrate CBR with RDBMS technologies are only one facet of a more general issue, i.e., the structural integration of CBR components into a more general software architecture, and in particular web-based architectures. That problem is illustrated by the following issues:

- **Legacy database reuse**. Large data repositories in the form of relational databases usually are the basic sources of information that cases should only reference and not copy. For instance, in a travel recommendation system alternative travel products (e.g. hotels, cruises, etc.) are described in content management systems developed by a network of suppliers that distributes and repackage those items.
- **Support case exchange**. Cases or case components must be easily exchanged between cooperating systems. This is a must for distributed architectures as those based on the web or on the cooperating agents framework. In the travel industry a customer case is usually cooperatively managed by different service providers, where each of them is responsible only for a component of the global product.
- **Support partial case exploitation**. A case tends to be a net of distributed chunks of information as an hypertext document or a view over multiple data sources (distributed databases). Each chunk can be stored on a specialized server and a reasoning task can involve only a part of the complete case.
- **Blend problem-solving with information retrieval**. Case-Based Reasoning systems more and more often integrate typical information retrieval functions like those supported by a standard query language, data mining models construction and recently also OLAP (Online Analytical Processing) like advanced aggregation and comparison functions [6]. Therefore data must be shared among these components and data transformation should be limited at the base minimum.

These are challenging issues for the CBR community and there is a growing interest on such themes (for a discussion of these works the reader is referred to Section 5). In this paper we present a ``middleware" component based on Java and XML that addresses the architectural requirements quoted above. This component is largely based on a new standardization initiative whose goal is to define a reference API for XML database interaction and data manipulation technologies. Our CBR component is built upon this generic module as an extension of the proposed architecture. It gives support for classical similarity-based retrieval (over semistructured data in the form of XML documents) and adds some special support for query relaxation based on new indexing technologies [6].

We illustrate the application of such a framework in a case study, a recommendation system for traveler destination selection [14]. The information source is a complex already existent relational database over which cases are built as application specific views and are modeled as XML (data centric) documents. This system is under development at eCTRL (Electronic Commerce and Tourism Research Lab) in collaboration with the organization for the tourism promotion of Trentino, and with other research and industry partners.

2. XML and Databases

XML (eXtensible Markup Language [3]), among other important functions, has been proposed since its introduction, as a mean for exchanging structured information between applications. XML enables the interchange of structured data in an easily extensible format that can be validated against a data definition schema. With XML heterogeneous systems could adopt a single industry-wide interchange format that serves as the single output format for all exporting systems and the single input format for all importing systems.

Nowadays, the ``integration'' capability of XML have been pushed forward. Mediator systems assume that a mediated schema[1] is a ``view'' over a heterogeneous data repository that can be physically distributed and implemented with many alternative technologies [9,12]. To support this application framework there have been proposed new technologies, namely ``XML databases'' [2] and XML query languages [7]. Today there is a number of tools that supports these integration and database functions (e.g. eXcelon, Tamino, Nimble). Figure 1 illustrates the basic architecture of a mediator system. A client accesses an XML based server (XML database) that exports data views in the form of some schemas. The client can send a query over a view that is executed on the server retrieving data from many physical data containers by means of a mapping software layer.

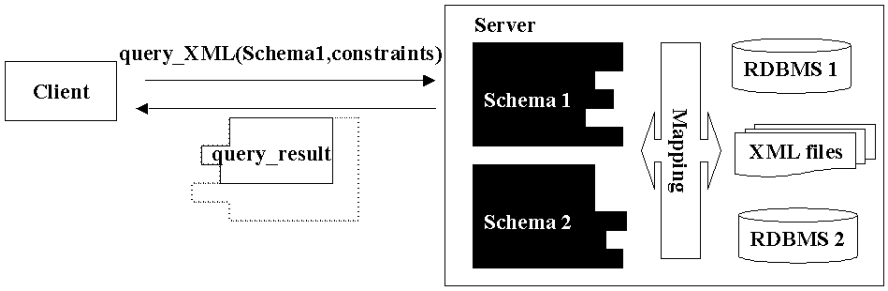

Figure 1: XML view-based integration approach

Figure 1 illustrates a possible implementation of an XML database, i.e., as a mapping (middleware) component towards a traditional RDBMS or a file system. In fact, there are essentially two categories of XML databases according to [2]:Middleware and Native XML database. The first are software components that can be called from a client application to transfer data between XML documents and databases. This can even be provided by databases with extensions for transferring data between XML documents and themselves. The second type of products stores XML in "native" form, i.e., maintaining the structure of XML documents and are said to be optimized to the XML format. Irrespective of the implementation, the basic functions that such components must implement are very basic and well known: to retrieve a document or a set of documents (resources) from the database using a known ID or a query condition; to store a new document; to remove and update an existing document.

[1] This may be defined using alternative schema languages as DTD or XML Schema.

One of the most serious limitation of the current XML database products is the lack of a standard API for system integration. In other words there is nothing similar to ODBC (or JDBC) for a standard interaction with the data sources. For that reason, the engineer that would like to exploit an XML database in his application architecture must carefully, and risky, choose between alternative products that could be hardly replaced during the project evolution.

But the situation is changing. XML:DB (http://www.xmldb.org) is an industry initiative formed by SMB GmbH, the dbXML Group L.L.C and the OpenHealth Care Group, that provides a community for collaborative development of specifications for XML databases and data manipulation technologies. Along with each specification an open source reference implementation will be developed to validate the ideas put forth in the specification and to more rapidly drive acceptance of the specification in real products.

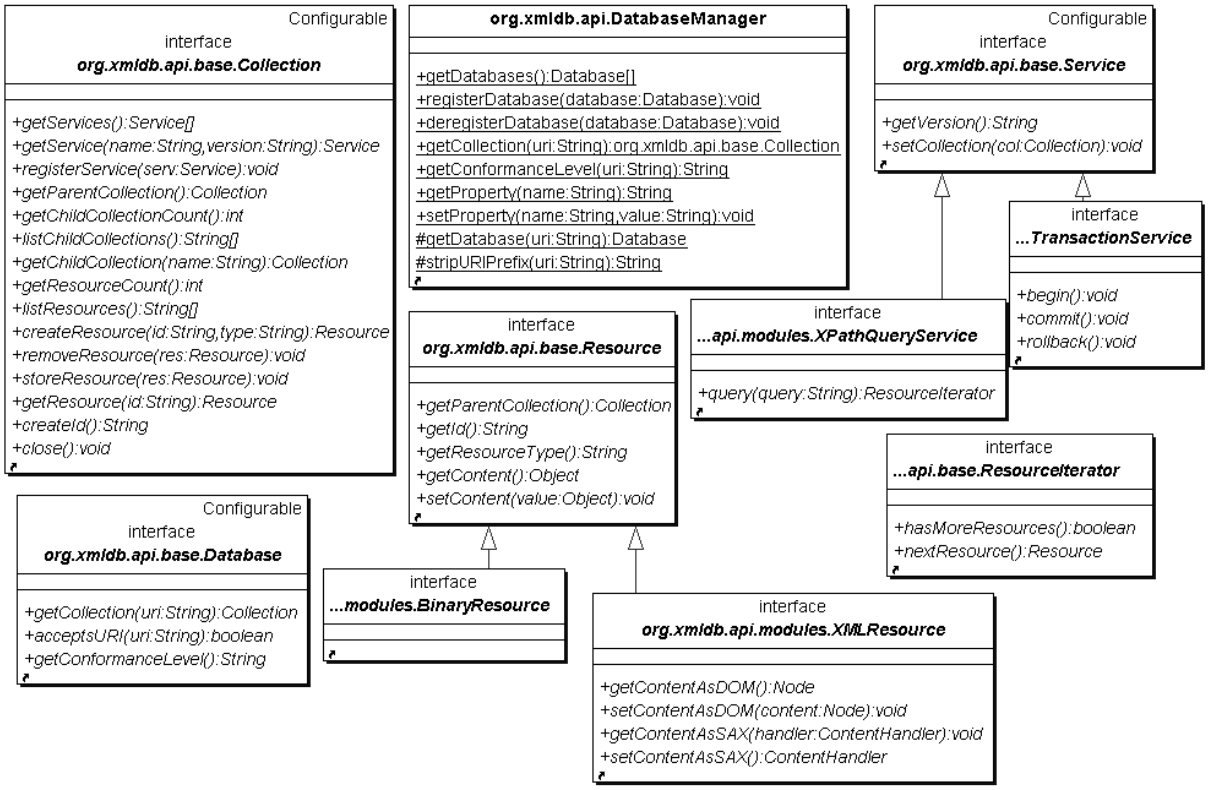

Figure 2: XML:DB main classes

Figure 2 shows the main interfaces proposed in the XML:DB API Base module[2]. We will briefly describe those interfaces referring the interested reader to quoted web site for a more detailed description. Note that we shall use teletype font for classes and methods.

[2] This figure refers to the May 7th 2001 working draft.

- `DatabaseManager` is the entry point for the API. It is the control point that provides access to Collections. Using the `DatabaseManager` a client application can obtain all `Database` instances and `Collection` objects known to the `DatabaseManager`.
- `Database` represents a driver for a particular XML database. It enables to get a `Collection` instance, that is located using a URI specific to the underlying database.
- `Collection` represents a collection of Resource objects stored within an XML database. `Collection` is the primary mechanism for getting access to resources and services that operate on those resources. A `Collection` can be queried to obtain all the (query) services supported or the `Resource` objects contained.
- `Resource` is an abstract container wrapper for data stored in an XML database. It exists because there are several different ways of accessing data in XML databases e.g. text, DOM (http://www.w3.org/DOM/), SAX (http://www.megginson.com/SAX/) and binary content.
- `ResourceIterator` provides an iterator over a set of `Resource` objects.
- `Service` provides a mechanism to enable specific processing to occur within the context of a collection. It is extension mechanism for the API to enable it to handle situations not foreseen at the time of the APIs creation. The `Service` interface just defines the basic control methods that are necessary to register and use a service. `Service` implementations will define their own detailed interfaces that will vary from service to service. For instance, `XPathQueryService` provides a service to query a collection using XPath expressions. The query is applied across all documents in the collection and a `ResourceIterator` is returned containing the results.

A module implementing this interface can provide a basic uniform wrapper over both XML (textual) and Relational data sources. We have implemented a (Java) driver for a Database that maps relational data, which are accessed through the JDBC API, to the XML format. In the next Section we will show how this component can be extended to support additional type of query that are proper of a CBR system and some additional OLAP functions.

3. CBR on Top of XML Databases

In this Section we describe how to extend the XML:DB base module with two services. The first is devoted to the computation of a ``classical'' range query extended with some methods supporting query refining, the second is devoted to the computation of a similarity query. Next, we will show how a `Collection` instance can be viewed as a Case-Base. Besides, we are assuming that our XML documents are data-centric, i.e., are characterized by fairly regular structure [2]. In this case, fine-grained data (that is, the smallest independent unit of data is at the level of a PCDATA-only element or an attribute), and no mixed content is present.

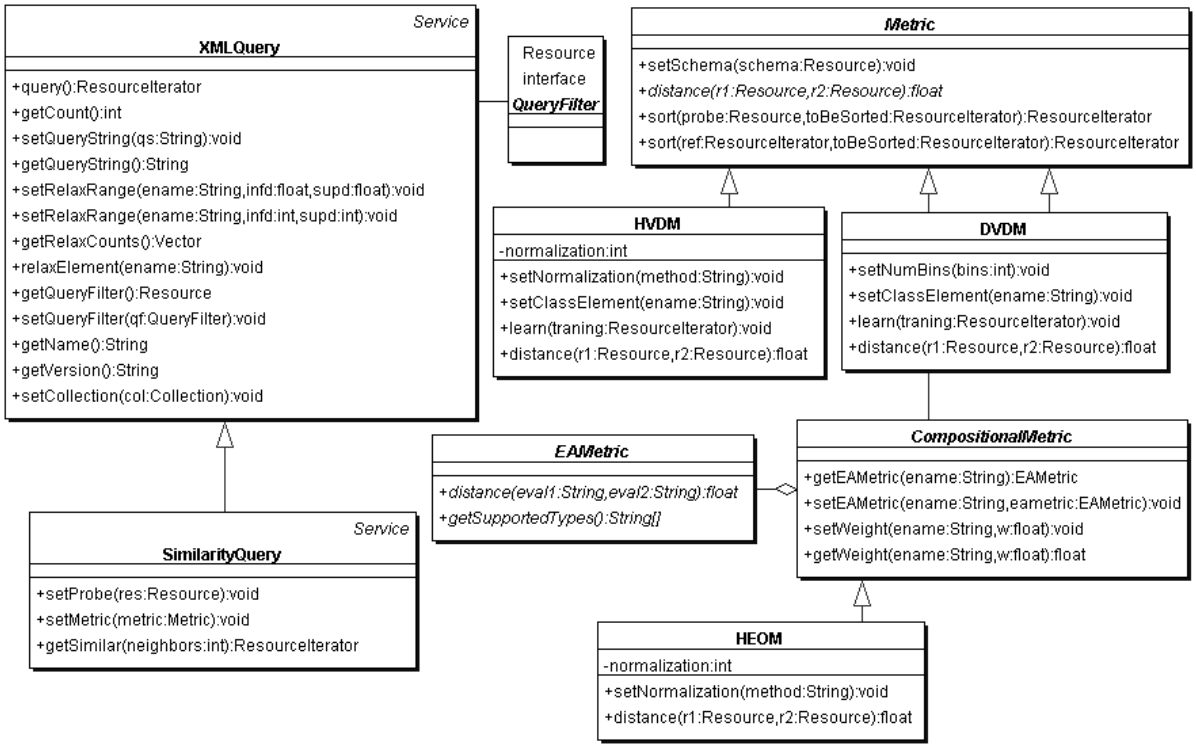

Figure 3: Classes for similarity-based retrieval

Figure 3 shows the main classes of our extension, that are here described:

- `XMLQuery` This class models a query object and collects functions for executing and refining a range query over a ``data centric'' XML database. An `XMLQuery` is generated by a `Collection` (as for the other services) calling the method `getService()`.
  - `setQueryString(String queryString)` This method set the definition of the query. The query language (not shown here) is very simple, it enables to express conjunctions of range conditions over element or attribute values[3]. As an example of query string the reader may consider ``cost > 90 and cost < 100 and parking='yes' ''. In this example the ``cost'' element is constrained to be in the [90,100] range and the boolean element ``parking'' must be 'yes'.
  - `query()` With a call to this method the query is executed on the `Collection` from wich the `XMLQuery` object was obtained.
  - `relaxElement(String ename)` In case the query returns no results it is possible to revise the query definition relaxing the query conditions over the elements that are constrained in the `queryString` (as it was set by the corresponding `setQueryString()` method). This is done with a call to `relaxElement(String ename)`. It is also possible to set the relaxation degree with a call to `setRelaxRange()`.

[3] The query language syntax is derived from SQL and is meant to interact with ``data centric'' document. This language is not show here for lack of space. The XPathQueryService, in the XML:DB core module is more suitable for ``document centric'' xml documents.

- `getRelaxCounts()` Deciding what are the elements in the query strings that caused a no-result effect could be complex in a real system. This method indicates for each element in the query string how many items could be retrieved by an updated query in which the condition on that element is relaxed. For instance if setting the query string to ``cost > 90 and cost < 100 and parking='yes' '' caused no-results, then a call to `getRelaxCounts()` returns a vector of this type *((cost, 32), (parking, 0))*. This means that relaxing the constraint on the ``cost'' element will produce a query that returns 32 items, whereas relaxing the condition on ``parking'' does not change the situation. In this way, a client application can probe in the neighborhood of the `Resource` objects contained in a result set. This function can be exploited to build a ``conversational'' support to query refining by indicating to the user the culprit of the ``no solution'' result and suggesting the most effective relaxation [10,1].

- `SimilarityQuery`This service specializes the `XMLQuery`. Basically, a similarity query retrieves from a `Collection` those resources that are most similar to a given `Resource` that is defined with a call to the `setProbe()` method. Similarity between two `Resource` objects is computed with a `Metric` object as the inverse of a distance function. If a `Resource` can be descibed as an array of features, and $x = (x_1,\ldots, x_n)$, $y = (y_1,\ldots, y_n)$ are two `Resource` objects, then the distance between these two resources is given by $d(x,y) = \sum_{i=1}^{n} w_i\, d_i(x_i,y_i)$, where w_i are positive weight numbers that are used to balance the relative importance of the features, and d_i are metric distances on the features (numeric or symbolic) [18]. Searching in a `Collection` those cases more similar to a given probe is computed by a nearest neighbor algorithm implementation (`getSimilar()` method). Moreover, the `SimilarityQuery` class implements the basic idea that a similarity-query must be executed after an `XMLQuery`, i.e., resources similar to a probe are searched in the results set of a classical range query. This is motivated by two reasons. First, similarity evaluation is typically performed on the middleware level, not where data are stored and the range query is executed (RDBMS)[4]. This implies that a similarity match must be executed for each resource in the result set, and those match computations must be reduced at the minimum possible level for computational reasons. Second, typical usage of similarity-based queries is for ranking of a set of potential interesting items among those that satisfy some logical constraints expressed in the queryString.
- `Metric` This interface provides the basic operations for comparing two resources, i.e., instance documents belonging to the same schema. As schema language we use XML Schema [8], primarly because of the added capability, with respect to DTD, to declare precise type information for element and attribute values. This class has a method for distance computation between two resources (`distance()`), and two `sort()` methods for ordering a `ResourceIterator` putting first the resources closer to a probe or to another set of resources. In Figure 3 are shown some examples of heterogeneous metrics, i.e., metrics that can deal with elements and attributes of different type (both numeric and symbolic) [19]. Moreover, this metrics extend the classical definitions, designed for a resource modelled as an array of features, because they operate on (XML) tree-like data structures [4,13].

[4] This can change in the future when RDBMS will natively support similarity-based queries.

In our extension of the XML:DB framework there is no need to introduce new classes for the case and case base concepts, in fact we have:

- **Case Base** Is a standard `Collection` for which some additional metadata information is provided, in two separate XML files.
- **Case** is a `Resource` in a Case Base `Collection`.

In order to view a Case Base as a `Collection` two metadata files are needed:

1. **XML Case Base Schema.** It is an XML schema file. `Resource` objects (cases) in the case base are constrained by this XML schema that will usually come from a standardization body (independent vocabulary). The schema specifies type information for attributes and elements and can be eventually extended, in another schema file, e.g., with the `redefine` element provided by the XML Schema language [8].

2. **Element and Attribute mapping and other metadata.** This is an XML file. Elements and Attributes in the `Resource` (case) Schema that belong to the case description must be listed in this metadata file and their usage in the CBR context must be specified. For instance, according to a classical distinction in CBR technolgy, for each feature it must be specified whether it belongs to the ``problem'' definition (the set of predictive features) or to the ``solution'' component (the set of predicted features) of the case. Other metadata are included as well, for instance: the existence of indexes or the selection of some special metrics for case comparison[5].

```
<?xml version=''1.0''?>
<ht:Hotel xmlns:ht = "http://hotelstandard.org/ht
"http://ectrl.itc.it/cbr">
         Type=''Hotel''>
  <Position>
    <Latitude>41.884</Latitude>
    <Longitude>12.498</Longitude>
  </Position>
  <HotelCategory>4</HotelCategory>
  <Name>Visconti Palace</Name>
   ...
  <UserClass>3</UserClass>
</ht:Hotel>
```

```
<?xml version=''1.0''?>
<cbr:caseBaseDescription xmlns:cbr =

  <name>Hotels</name>
<caseBaseCollection>xmldb:jdbc:odbc:hotels</caseBaseCollection>
  <caseElement>Hotel</caseElement>
  <feature>
    <name>Type</name>
    <isInProblem>y</isInProblem>
  </feature>
  <feature>
    <name>Latitude</name>
    <isInProblem>y</isInProblem>
  </feature>
...
  <feature>
    <name>UserClass</name>
    <isInProblem>n</isInProblem>
  </feature>
  <metric>
    <name>HEOM</name>
    <normalization>range</normalization>
  </metric>
</cbr:caseBaseDescription>
```

Figure 4: An XML Hotel case example and the CBR metadata for that case

[5] The XML schema of this document is not included here for lack of space.

Figure 4 show a very simple example. On the left, it is shown an hypothetical case that model an hotel that is described according to an existent XML schema tagging language (not shown here). On the right, it is illustrated an excerpt from the XML metadata file that maps attribute and element names to features of the case. Here, for instance, the `<caseBaseCollection>` element specifies the URI of the `Collection` that stores the Case Base. The `<isInProblem>` element states if a feature, with given `<name>` in the XML case description is either part of the problem or of the solution component. Finally the `<metric>` element declares the distance metric to be used in the similarity computation.

In Summary, a `Collection` of `Resource` objects, which can be physically implemented as a view over a relational database or as collection of XML documents, becomes a case base provided that a minimal set of metadata information is added. This approach enables to integrate seamless CBR functions in a more ``conventional'' software application.

4. Case Study: Travel Planning

In this section we briefly sketch an application of the proposed framework to the implementation of an intelligent recommendation system aimed at supporting a leisure traveler in the task of selecting a tourist destination. The system enables the user to build up his own personalized travel by aggregating elementary items (locations, services and activities). Case-Based Reasoning techniques enable the user to browse a repository of past travels or a catalogue of services and rank elementary items included in a recommendation. The system integrates data and information originating from external, already existent, tourist portals exploiting the mediator architecture here described [14].

The central building block of this recommendation system is the TravelAsset resource (TA in short). This is can be a location, a tourism service or an activity performed during a certain timing. The second modeling assumption refers to a TravelPlan i.e. a coherent aggregation of TravelAssets chosen by the traveler to be ``consumed'' during the travel. The set of all TravelPlan objects composed by a user using the system plays the role of the main case base. But case-based reasoning techniques (retrieval) are used to search and select a single TravelAsset as well.

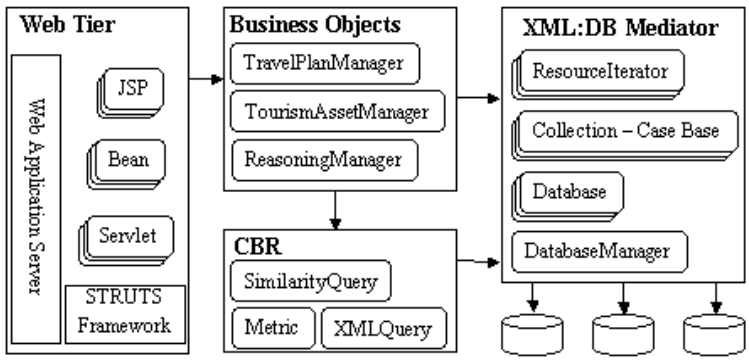

Figure 5: Architecture of the Recommendation System

Figure 5 depicts the general architecture of the recommendation system. The reference application model is the Java2 Enterprise Edition Architecture (www.java.sun.com/j2ee/). In the Web Tier we use Struts, an open source framework useful in building web applications with Java Servlet and JavaServer Pages (JSP) technology. Struts encourages application architectures based on the Model-View-Controller (MVC) design paradigm (http://jakarta.apache.org/struts/). The Business Object tier contains the TravelPlan and TravelAsset implementations as well as three manager objects that implement the main system functions (management of the plans and recommendation support). The CBR and XML:DB components implement the mediator architecture enabling to retrieve data from legacy databases and to model this information as case bases.

Regarding the recommendation functions, the user has two main modalities to obtain a destination recommendation. First, he can query a repository of TravelAsset objects. This is supported by classical and similarity based search and scoring. In this modality a TravelAsset is viewed as a case and repositories of various product/services play the role of Case Bases. Second, the user can search the repository of past suggested TravelPlan objects for travels made by other "similar" users or in conditions similar to those specified by query constraints or by a tentative (incomplete) probe TravelPlan that the user is composing.

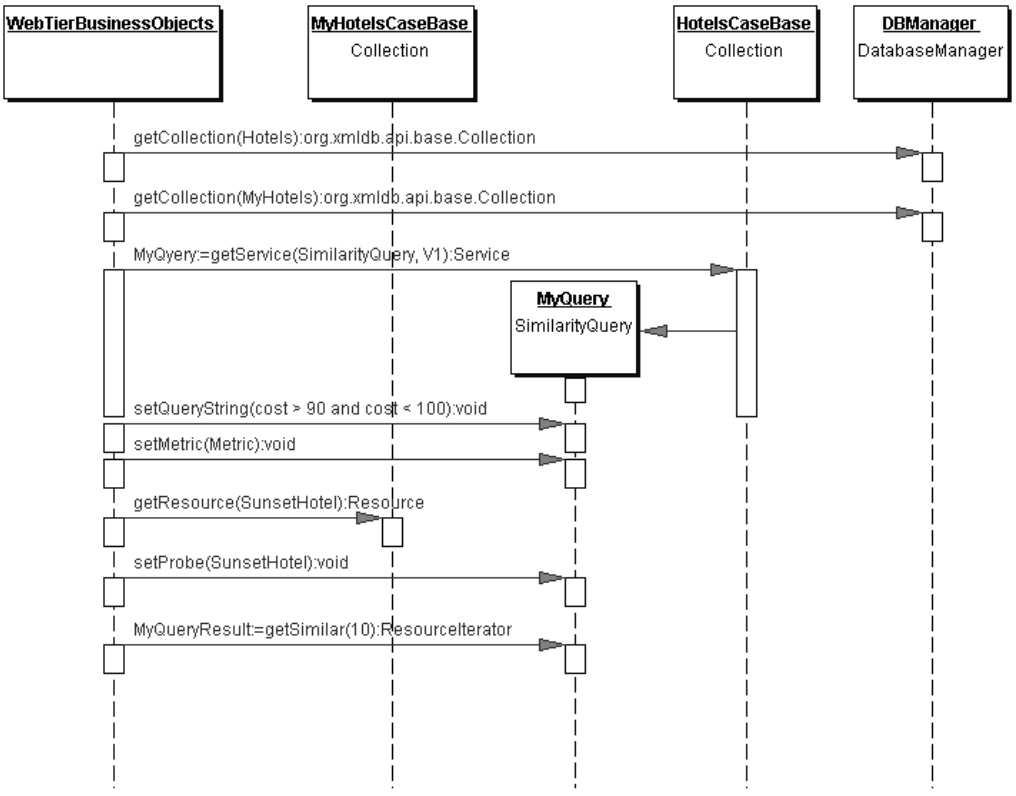

Figure 6: Similarity based retrieval and result scoring scenario

Figure 6 depicts a typical similarity based retrieval and case scoring scenario in a web application context. In this sequence diagram the details of the control component that manages the man/machine interaction (Web Tier and Business Objects) are not shown. For sake of

simplicity the flow seems to be generated by an abstract (not existent) WebTierBusinessObjects object.

1. `getCollection(Hotels)` A `Collection` of hotels is retrieved using a `DatabaseManager` object. This is the case base collection containing all the hotels described in the system.
2. `getCollection(MyHotels)` The users's collection of hotels is then retrieved from the `DatabaseManager` object.
3. `getService(SimilarityQuery, v1)` Using the `HotelsCaseBase` a new `SimilarityQuery` (service) is built.
4. `setQueryString(cost > 90 and cost <100)` A query string is set for the query object. This means that all the resources retrieved by similarity with respect to a given probe must satisfy the logical conditions expressed in the query string (here this is ``cost > 90 and cost < 110'').
5. `setMetric(Metric)` A new metric is built and assigned to the `SimilarityQuery` object.
6. `getResource(SunsetHotel)` An hotel in the personal case base of the user is retrieved. The underlying hypothesis is that this hotel has been liked by the user and now can be used as means to compare other hotels in the selection.
7. `setProbe(SunsetHotel)` The previously selected hotel is set as the probe of the similarity query.
8. `getSimilar(10)` 10 hotels similar to the probe are retrieved from that subset of cases in the `HotelsCaseBase` that satisfy the query condition set by the operation `setQueryString()`. At this point the physical storage system (here a RDBMS) is accessed. It executes the query expressed by the logical conditions and returns this to the similarity computation filter.

More complex scenario (not shown here for lack of space) are supported by the framework as well. What we would stress is that the additional burden due to CBR typical processing do not involve any modification of the data sources. Only the CBR services that extend the XML database framework, and some minimal metadata information must be implemented.

5. Related Work

The intersection of CBR and XML is becoming an interesting area of research. Shimazu [17] has been among the first to advocate the exploitation of XML in CBR application. XML was introduced first as a case representation language, i.e for structuring the knowledge in a case, and secondly because of the special support of XML for textual data. In the CARET/XML system each case is modeled with a case profile that contains: a) a link to an XML file with the full case description; b) a set of fields (and field values) extracted from the corresponding XML file. Case profiles are stored in a RDBMS. This is a interesting and early implementation of an XML to RDBMS (partial) mapping. The similarity computation is performed only on that part of the case that is stored in the RDBMS.

Sengupta et al. [16] propose a classification of CBR implementation into three types: Web-Based, Enterprise and Task-Based. The distinction between these three models is mainly due to the case representation language and case physical storage: Web-Based systems use XML,

Enterprise systems uses RDBMS and Task-Based use ad hoc representation languages and various storage systems. Our proposal tries to merge at least the web-based and the enterprise models, by providing an API which is independent from the physical storage system whether it be a native XML database or a RDBMS.

Hayes and Cunnigham [11] already argued that a case base document should be considered no differently from a document containing industry standard data. They propose to add markup from a CBR namespace to a document already tagged with an industry specific markup language (from another namespace). We believe that this solution introduces the big burden of re-tagging an XML datasource and requires to modify the original data. Differently, the approach proposed here, i.e., extending the original schema with those type information that were not present and adding an xml document with metadata information (role of the original tagging information and other CBR relevant data) introduces a very low additional work to convert a legacy XML data source into a full fledged case base.

6. Conclusion

This paper describes a work in progress in a new research laboratory (Electronic Commerce and Tourism Research Lab - http://ectrl.itc.it) devoted to the exploitation of enterprise data in a CBR system for travel recommendation.

The goal of this work is to develop a software framework for easing the implementation of case-based reasoning components that are seamless integrated in a enterprise architecture. The framework exploits recent standardization initiatives in the area of XML databases and mediator based architectures. We have shown how a legacy XML data source, which may be tagged with a standard vocabulary, can be easily reused as a case base with minimal additional efforts. An application of the proposed framework to the development of a case-based recommendation system for tourism travel planning, which is ongoing at our laboratory, is illustrated.

At the current stage we have delivered the system architecture and the component design of the major packages. We have started the development of the mediator, that is our core component and will represent the proof of concept of the technological solution proposed in this paper. We must however note that the approch shown here is still in an early development stage and some major CBR component is missing (revise and adaptation of the retrieved case).

8. References

[1] D. Aha and L. Breslow. Refining conversational case libraries. In *Case-Based Reasoning Research and Development, Proceedings of the 2nd International Conference on Case-Based Reasoning (ICCBR-97).* Springer-Verlag, 1997.

[2] R. Bourret. Xml and databases, 2000. Available at "http://www.rpbourret.com/".

[3] T. Bray, J. Paoli, C. M. Sperberg-McQueen, and E. Maler. Extensible markup language (xml) 1.0 (second edition). W3C Recommendation 6 October 2000, 2000.

[4] H. Bunke and B. Messmer. Similarity measures for structured representations. In S. Wess, K.-D. Althoff, and M. M. Richter, editors, *Topics in Case-Based Reasoning*, Kaiserslautern, Germany, 1994. Springer-Verlag.

[5] R. Burke. *Encyclopedia of Library and Information Science*, chapter Knowledge-based Recommender Systems. 2000. To appear.

[6] S. Chaudhuri and U. Dayal. An overview of data warehousing and olap technology. *SIGMOD Record*, 26(1):65-74, 1997.

[7] S. Cluet, A. Deutsch, D. Florescu, A. Levy, D. Maier, J. McHugh, J. Robie, D. Suciu, and J. Widom. Xml query languages: Experiences and exemplars. http://www.w3.org/1999/09/ql/docs/xquery.html.

[8] D. C. Fallside. Xml schema part 0: Primer. http://www.w3.org/TR/xmlschema-0/, March 2001. W3C Proposed Recommendation.

[9] D. Florescu, A. Levy, and A. Mendelzon. Database techniques for the world-wide web:a survey. In *Proceedings of the ACM Sigmod-98*. ACM press, 1998.

[10] M. H. Göker and C. A. Thomson. Personalized conversational case-based recommendation. In *Advances in case-based reasoning: 5th European workshop, EWCBR-2000, Trento, Italy, September 6-9, 2000: proceedings*. Springer-Verlag Inc., 2000.

[11] C. Hayes and P. Cunningham. Shaping a cbr view with xml. In K.-D. Althoff, B. R, and B. L.K., editors, *Case Based Reasoning Research and Development, Proceedings of the International Conference on Case Based Reasoning (ICCBR)*, Monastery Seeon, Germany, 1999. Springer Verlag.

[12] A. Y. Levy. The information manifold approach to data integration. *IEEE Intelligent Systems*, pages 12-16, september/october 1998.

[13] F. Ricci and L. Senter. Structured cases, trees and efficient retrieval. In B. Smyth and P. Cunningham, editors, *Advances in Case-Based Reasoning*, volume 1488 of *Lecture Notes in Computer Science*, pages 88-99. Springer Verlag, 1998.

[14] F. Ricci and H. Werthner. Case-based destination recommendations over an xml data repository. In *Enter2001*, Montreal, Canada, 24-27 April 2001.

[15] J. Schumaker and R. Bergman. An efficient approach to similarity-based retrieval on top of a relational database. In *Advances in case-based reasoning: 5th European workshop, EWCBR-2000, Trento, Italy, September 6-9, 2000: proceedings*. Springer-Verlag Inc., 2000.

[16] A. Sengupta, D. C. Wilson, and D. B. Leake. On constructing the right sort of cbr implementation. In *CBR workshop at IJCAI'99*, 1999.

[17] H. Shimazu. A textual case-based reasoning system using xml on the worl-wide web. In B. Smyth and P. Cunningham, editors, *Advances in Case-Based Reasoning*, volume 1488 of *Lecture Notes in Computer Science*, pages 274-285. Springer Verlag, 1998.

[18] D. Wettschereck, T. Mohri, and D. W. Aha. A review and empirical comparison of feature weighting methods for a class of lazy learning algorithms. *AI Review Journal*, 11:273-314, 1997.

[19] D. R. Wilson and T. R. Martinez. Improved heterogeneous distance functions. *Journal of Artificial Intelligence Research*, 11:1-34, 1997.