The Decomate II Current Awareness Service

Ferran Jorba[1]
Servei d'Informàtica
Universitat Autònoma de Barcelona
22 June 2000[2]

Abstract

The Decomate II Current Awareness Service is an extension of the standard search services of the system. This means that any of the Z39.50 databases available from Decomate II can be queried to extract new records that match the users interest profiles. This represents an interesting challenge, as the characteristics (fields and indexes) of those implementations are diverse. In this paper we will see how users can manage their profiles, how the results are presented on screen and also how Decomate II CAS implements different search strategies to extract new additions to the databases without getting any privileged access to them.

CAS design goals

The main purpose of the Decomate II system is to provide an unified access to the diverse sources of bibliographic information that libraries deal with. The library catalogue, the scientific bibliographies, remote journals at the publisher's web sites, cooperative resources like RePEc, etc., each of them have their own native interface, but the Decomate II system allows to access them from a single entry point. If a single interface is a good thing for searching, a single alerting system is a logical extension for it. That's the task of the Current Awareness Service.

At the same time, the CAS functionality should be seamlessly integrated into the main system, and perceived by the end user as part of the same whole, not as an add-on. And not only from the end-user point of view: internally, the Decomate II CAS keeps an internal coherency with other components of the system and talks the same XREP[3] protocol. The interfaces with them are clear and clean: this means that it can run either on the same computer than the rest of the system, or on a different one.

User interface and functionality

The Decomate II CAS notifies subscribed users about new bibliographic records that match a query for an user-selected list of databases. This query, together with other user-defined delivery preferences, is called an *interest profile*. A new interest profile can be created by the user after performing a search over one or more databases. The screen

[1] Author's address: Ferran.Jorba@uab.es.
[2] Presentation given at the conference *Logged into Economics : an Assessment of the European Digital Library Decomate II*, Barcelona (Spain), 22nd-23rd June, 2000.
[3] XML Request and Response Protocol, explained elsewhere in this conference.

that shows the results of the search also has a button to save the query, and define his or her choices: (s)he can set a name, frequency, email address, etc.

For each new interest profile created, the system sends a welcome message via email. At the frequency chosen by the user, the system searches the selected databases for new bibliographic records that match the query, and, if requested, it sends an email notification. At the same time, it creates a virtual personalised journal with those records. For each run, a new issue of this journal is created, which can be browsed via the standard Decomate II Web interface, for which only the "subscriber" has access to. Those records are locally stored in order to guarantee both a fast access and as a safeguard against additions or changes in the remote databases. Of course, an interest profile can also be modified or deleted by the end user at any time.

The system also tries to give flexibility and maintenance independence to the system administrator: most of the values can be parametrised: the number of issues stored per interest profile, the number of bibliographic records per issue, etc. It also deletes expired users: in the university community people come and go, so the CAS system takes care, via the Authentication Broker, to check user validity and permissions to the databases (as they may change, or users may change status), and warns him or her via email if they are wrong. After this notification has been sent and a number of grace days has passed (another parameter yet), the profiles and issues for those expired users are automatically removed.

CAS components

The CAS consists in three parts: the CAS Interactive Session, the CAS Server and the CAS Robot.

1. The CAS Interactive session handles users requests and provides the logic for presenting and updating the screens. As a clear user-interface task, it is handled by the Broker's functionality[4].
2. The CAS Server handles CAS Interactive Session requests and translates them from XREP to SQL and back. In other words, it is an XREP server and a SQL client.
3. The CAS Robot does the bulk of the heavy work. It checks all the available profiles, calculates frequencies, decides whether the queries have to be performed and requests the broker to execute them. It then filters new bibliographic records, keeps the results, updates the issues and sends the results via email to the users.

Strategies to discover new bibliographic records

Most of the bibliographic alerting services are part of their data-loading procedures, so they get those new records and distribute them to the subscribed users. There are plenty of examples of this model that work well on a limited and controlled data[5], not necessarily bibliographic. The opposite one, where a service "watch" events —in a non-

[4] And due to that, it has been implemented by the Broker team at Tilburg University.

[5] For example, the table of contents email alerting service of the Consorci de Biblioteques Universitàries de Catalunya (http://sumaris.cbuc.es).

privileged way— and notify users about changes is also popular on the Internet[6]. But none of them allows the user to set a specific frequency to be notified about; rather, it is an event-based system where the timing is determined by the observable event classes, and system sends notifications to the subscribers[7] as they happen, or at a fixed frequency[8].

In Decomate II, the scenario is quite different: there is an unknown set of requests (interest profiles), over a distributed collection of bibliographic data (external Z39.50 repositories, with no privileged access to them), to be notified at different frequencies (user chosen) and possibly delivered in different formats. Moreover, those Z39.50-accessible databases have different attributes, indexes and characteristics.

So, as the databases are diverse, we thought we'd better devise a set of different strategies to deal with those situations. We looked at the most obvious indexes to identify new records: a date field[9] first, and a record id second. A Z39.50 database with those indexes should be a good one. Really? How good is a date index if we cannot search with a "greater than" comparison? The same can be said about the record id field. Most of the databases have at least one of those fields indexed, but it is not so common to allow the "greater than" comparison. Others do not even have those fields indexed, and also a few of them do not even have them. For the last case, we had to imagine a way to build an unique identification for those records (we called it a calculatedID). We ordered them from better to worse, and we came out with those strategies:

A. searchByDate
B. searchByRecordID
C. compareLastDate
D. compareLastRecordID
E. checkStoredRecordIDs
F. checkStoredCalculatedIDs

Later on, well advanced in the implementation phase, we found out that with a judicious use of parameters, we could reduce them into four, grouping C with D and E with F, giving this shorter list[10]:

1. searchByDate: when the date field can be searched by "greater than".
2. searchByRecordID: when the record id field can be searched by "greater than".
3. compareLastID: when there is an identification field that belongs to an ordered set, so we can compare it locally with a "greater than" with the last one retrieved.

---

[6] Like Netmind's Mind-it (http://mindit.netmind.com), which monitors page changes and alerts subscribed users about them.

[7] For a good discussion of different models of alerting services, see Annika Hinze and Daniel Faensen, *A Unified Model of Internet Scale Alerting Services* (see bibliography). Section 2 describes event notification services. An interesting insight the authors provide of the digital library scenario is that "Objects of interest are unknown at the time of profile definition and usually come into existence later" (section 3.1.2).

[8] Like the weekly ISI "Discovery Agent" (http://www.isinet.com/products/alertsvc/daprod.html).

[9] The more common word "field" is used here, although in proper Z39.50 terminology they are called "use attribute".

[10] We didn't group A with B due to the well known difficulties to deal with dates and date formats, specially when used as a search field.

4. checkStoredIDs: when there is no id belonging to an ordered set that can be compared with a “greater than” operation, each of the incoming records must be matched against all records found so far, using an “exclusive or” operation. If the database doesn’t provide a proper record id field, a unique identification has to be generated, based on a combination of fields that creates non-repeteable keys. An MD5 signature can be optionally calculated when the string is too long to fit in CAS tables.

Those strategies are stored in the Decomate II metadata, where database information is stored and administered. For each database, a CASSearchStrategy entry must be added, for example:

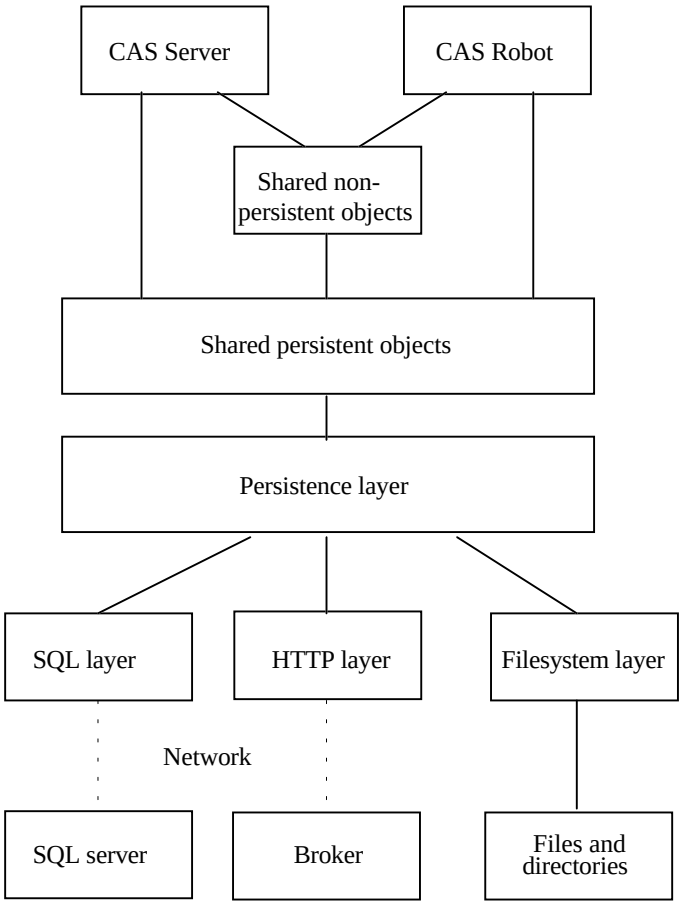
```
<CASSearchStrategy>
  <method>searchByDate</method>
  <searchField>ld</searchField>
  <dateFormat>yyyy.mm.dd</dateFormat>
</CASSearchStrategy>

<CASSearchStrategy>
  <method>searchByRecordID</method>
  <searchField>id</searchField>
</CASSearchStrategy>

<CASSearchStrategy>
  <method>compareLastID</method>
  <keyTag>RECORDID</keyTag>
</CASSearchStrategy>

<CASSearchStrategy>
  <method>checkStoredIDs</method>
  <keyTag>ISSN</keyTag>
  <keyTag>COLLATION</keyTag>
  <shortenKeyMethod>MD5</shortenKeyMethod>
</CASSearchStrategy>
```

We know of at least one system that is not able to fit here: there is one well known bibliographic service that doesn’t return more than 200 records via Z39.50. We understand, however, that this is a faulty implementation that has to be fixed.

Architectural model and implementation

CAS is a two-head beast. Both the CAS Server and the CAS Robot share the same objects and their behaviour. For example, the central object of the whole CAS system is the interest profile, and all the work is done around it, both by the Server and the Robot. There are also common objects like personalised journal issues, bibliographic records, parameters, etc. Thus, it was a logical decision to build an object oriented system that reuses the same objects and guarantees their coherent behaviour and an easier developer’s maintenance.

Those main objects (interest profiles, issues, bibliographic records, etc.) are persistent[11], that is, they can be stored and retrieved among different sessions. We had to choose a proper persistence repository for them. The primary *lingua franca* that Decomate II speaks to the outer world is Z39.50, and it makes sense to make CAS bibliographic data Z39.50-accessible. This allow the reuse of the Broker+MPS combo functionality for CAS journals bibliographic retrieval and presentation functionality. However, Z39.50 systems usually don't provide immediate indexing[12] or transaction control. Due to that, a more traditional and business-oriented relational database system is used for storing interest profiles and other non-bibliographic information. This further complicates the persistence layer in our system, that has interfaces with SQL, the filesystem and the Broker. In the later one, the Broker mediates with the Z39.50 functionality and the Authorisation Broker using standard HTTP. So, the CAS Server is an XREP server and a SQL client, and the CAS Robot is both an SQL and HTTP client. A graphical representation of the object hierarchy can be depicted this way:



CAS object hierarchy

[11] There is a large bibliography on object persistence, specially about the so-called *impedance-mismatch* between object-oriented programming and relational databases. See a selected bibliography at the end of this article.

[12] This is not a protocol fault or responsability, but rather than Z39.50 systems are, in general, build around free-text indexing systems over flat files. Moreover, database updates over Z39.50 are a relatively recent feature of the protocol (in version 3, 1995), and they are optional, in the so called *Extended Services*.

Conclusion

Decomate II CAS service is a generic system that allows a unified alerting service about new bibliographic entries from a distributed set of databases, without using any privileged access to them. It is based on the existing Decomate II infrastructure (Broker, MPS and Authorisation Broker) to send to the authorised users notifications about new publications that fit their interests, and to create personalised journals with those records. Almost any Z39.50 database accessible from Decomate II can be configured to deliver new records, selecting the right strategy and choosing the appropriate values.

A system like this, where there are factors beyond the developers control, like time (usually measured in weeks or even months) or the variety of Z39.50 implementations out there, make it quite difficult to test it properly. Due to that, it needs a long experimental period to settle it down and mature. However, we believe that we have build a solid and scalable system that allows further refinement in the future.

Acknowledgements

The Decomate II CAS has been a collaborative work of many people, who have enriched it with their suggestions and feedback. I would like to thank all the Decomate II partners for it, specially my colleagues at UAB Angels Sales, Laura Permanyer, Nuria Gallart and Carme Ribera (the latter no longer at UAB) and the project team at Tilburg University: Thomas Place, Hans van den Dool, Roel de Cock and Joost Dijkstra (who is no longer at TU), and of course the always needed testing and feedback from the teams at LSE and EUI.

Bibliography

Ambler, Scott W., *Building Object Applications That Work*, Cambridge, etc. : Cambridge University Press : SIGS Books, 1998.

—— *The Design of a Robust Persistence Layer For Relational Databases*, 16 August 1998 version (http://www.ambysoft.com/persistenceLayer.pdf).

—— *Mapping Object to Relational Databases*, 26 February 1999 version (http://www.ambysoft.com/mappingObjects.pdf).

Brown, Kyle, Bruce G. Whitenack, *Crossing Chasms : a Pattern Language for Object-RDBMS Integration : the Static Patterns*, Knowledge Systems Corporation, 1998 (http://www.ksccary.com/Articles/ObjectRDBMSPattern/ObjectRDBMSPattern.htm).

Buschmann, Frank, Regine Meunier, Hans Rohnert, Peter Sommerlad, Michael Stal, *Pattern Oriented Software Architecture : A System of Patterns*, Chichester : Wiley, 1996.

Hinze, Annika, Daniel Faensen, *A Unified Model of Internet Scale Alerting Services*, 1999 (http://www.inf.fu-berlin.de/~hinze/projects/publications/TR-B-99-15/).

Heinckiens, Peter M., *Building Scalable Database Applications : Object-Oriented Design, Architectures and Implementations*, Reading, Mass. : Addison-Wesley, 1988.

Object Matter, Inc., *Object Relational Mapping Strategies*, 1988 (http://www.objectmatter.com/vbsf/docs/maptool/ormapping.html).

Russell, Mark L., *Foundations of Object Relational Mapping*, v0.2 [mlf-970703], 15 July 1997 (http://www.chimu.com/publications/objectRelational/).

Salo, Timo, Justin Hill, Scott Rich, Chuck Bridgham, Daniel Berg, "Object Persistence : Beyond Serialization", in *Dr. Dobbs Journal*, vol. 24, n. 5 (May 1999), p. 19-33.

Turner, Paul, Arthur M. Keller, *Reflections on Object-Relational Applications* September 1, 1995 (http://www-db.stanford.edu/pub/keller/1995/reflections-object-relational.pdf).

Yoder, Joseph W., Ralph E. Johnson, Quince D. Wilson, *Connecting Business Objects to Relational Databases*, 1998 (http://jerry.cs.uiuc.edu/~plop/plop98/final_submissions/P51.pdf).