Software Components Retrieval Through Mediators and Web Search*

Robson P. de Souza[1], Marcelo N. Costa[1], Regina M.M. Braga[1,2]
Marta Mattoso[1] and Cláudia M. L. Werner[1]

{pinheiro, mcosta, regina, marta, werner}@cos.ufrj.br

[1]Department of Computer Science - COPPE/UFRJ
PO Box: 68511, Rio de Janeiro, RJ, Brazil, Zip Code: 21945-970
Telephone: +55+21+25628694, Fax: +55+21+2290-6626

[2]Department of Computer Science – CTU/UFJF
Benjamin Constant, 790, Juiz de Fora, MG, Brazil, Zip Code: 36015-400
Telephone: +55+32+3229-3800, Fax: +55+32+3229-3900

Abstract

Component Based Development (CBD) aims at constructing software through the integration, using interfaces and contracts, between pre-existing components.

The main goal of this work is to provide access to component that can be published at the Web, retrieved, and reused in all phases of an application development within a given domain. We present an architecture for software components reuse by using a mediation layer that integrates the semantics of Web components with previously registered components from a virtual library of components. In our architecture, components are described through XML documents and published by local repositories or remote servers. The innovative aspect of our proposal is the combination of mediators and software agents for reusable component retrieval within a Domain Engineering context. Mediators can represent application domain as well as integrate the description of domain related components. Queries can be issued to the mediation layer and processed by the GOA Object Server, which presents the query results as a list of suggested components along with its repository link in XML. Software agents are responsible for web component discovery and filtering. Techniques such as user models (profiles), and recommendations are used for presenting a ranked list of links. Finally, resulting links from mediators and web post-processed results are combined and presented to the user.

Keywords: *Component Retrieval, Component Based Development, Domain Engineering.*

1 Introduction

Component Based Development (CBD) [1] aims at constructing software through the inter-relationship between preexisting components. CBD reduces the complexity, as well as costs of software development, through the reuse of exhaustively tested components. The main goal of a software reuse environment is to provide access to components that can be reused in all phases of an application development within a given domain. Thus we are concerned with software components in general, not only with code, but also diagrams, use cases, models and other documents involved in the software development life cycle.

The Internet is the natural source of potentially reusable components. However, finding an adequate component involves searching among heterogeneous descriptions of components within a broad search space. Basically there are two approaches for component search: (i) Web search based on components interface; and (ii) search over a library of components that provides a semantic description of the components and their storage. Most works concerned with finding adequate preexisting components adopt either the Web approach [2, 3, 4] or the library approach [5, 6, 7].

The problem with the Web approach is that components can be published and described in many heterogeneous ways and a keyword-based search can be very generic. Also, the semantics represented by the components interface is very poor for reaching a useful component within a certain domain application. The library approach does not present these disadvantages but is restricted to previously and locally stored components. In this paper, we describe an architecture that combines the Web with the library approach. By using domain engineering with mediators we integrate the semantics of Web components with previously registered components from a virtual library of components. The definition of the semantics is encapsulated in

* This work was partially financed by Faperj, CNPq and CAPES.

mediators based on information from a specific application domain. This architecture comprises three main layers: (i) published components; (ii) mediation layer with component semantics and ontology services for published components, named *ComPublish*; (iii) agent layer that searches and filters retrieved components from the Web as well as from previously registered/published components, named *CompAgent*.

These services are part of a software reuse environment, named *Odyssey* [8], which aims at providing support for the development and reuse of components in all phases of software construction. The solutions we present here are implemented within *CompAgent* and *ComPublish* as an extension of the *Odyssey* search engine [9] to address the search and publication of components on the Internet. Our work was motivated by the Municipal Legislative project that was conducted at the Municipal Legislative House of Representatives from Rio de Janeiro (CMRJ). There are several applications that can benefit from reusable information within the legislative and related domains, such as judiciary and criminal domains. Our users are not specialists in these latter domains, only in the legislative. Therefore, reuse should be fostered on the legislative domain, but related domains should also be suggested to the non-specialist user through a specific relationship.

This work is organized as follows. In Section 2 we discuss our approach in comparision to related works from the technical literature. The publication of components with mediation services, the modules and services of the integration layer, is presented in Section 3. The several agents that filter and present potential components for software reuse are discussed in Section 4. Section 5 illustrates the application of our services in the CMRJ project. Section 6 presents our final remarks.

2 Related Works

The work presented by Seacord, Hissan and Wallnau [2] describes a search engine for the retrieval of reusable code components in the Agora system, such as JavaBeans and CORBA components. The Agora system uses an introspection mechanism for registering code components, through its interface. Agora search is Web based and searches only on component interfaces, covering solely the component connectiveness problem. The Agora system only deals with code components whereas our approach is concerned with domain information in all abstraction levels, including code components. Moreover, new information is always associated to domain terms within a given domain ontology [10], improving its accessibility and reuse in our architecture.

Another interesting work is found in [3]. ComponentSource is a web repository that provides services to buy, sell and develop components. The problem when searching components with ComponentSource is that components are described in a generic way, i.e., you cannot search for components based on domain functionalities. Thus, if a user wants to buy components related to the financial domain, there is not a search mechanism to restrict the search range. Moreover, ComponentSource only deals with code components like JavaBeans and CORBA components.

Regarding reuse library interoperability an important work to mention is the RIG initiative [5]. The idea of interoperability of asset libraries is based on the storage of domain information in several databases. These databases are static and based on one global model. RIG lacks a more effective search engine that provides searches based on domain concepts and filtering of relevant information, and with Internet access. Our approach uses the mediation technology with specific domain ontologies to integrate different software components data sources.

Ye and Fischer [7] present an approach that provides an active repository for components. Their work focuses on active delivery of reuse information, helping the reuse of components that developers were not aware. In this latter aspect, it is similar to our approach. Our component retrieval system provides this functionality too, since it accesses components from other domains based on semantic similarity. The active repository functionality, although not described in this paper, is also present in our previous work [11]. One aspect that is different in our proposal is the retrieval of distributed information, which is not mentioned in the work of Ye and Fischer.

3 Publication of Components

ComPublish is an architecture that aims at publishing software components and their related artifacts, such as models, diagrams, source code and other documents. Besides publishing components, *ComPublish* also provides a uniform view of components that belong to the same application domain. The main services of *ComPublish* are [12]: (i) to describe components based on domain information; (ii) to integrate this description with the semantics of other published components from the same domain; (iii) to provide search mechanisms over the published components; and (iv) to store and retrieve software components.

Figure 1 shows the architecture of *ComPublish*. *ComPublish* users interact through the Service Manager (SM) that provides a CORBA interface to list available domains (mediators), to access all related mediators, and to download a stored component. The integration of component descriptions is mapped on mediators. The Object Server GOA [13] stores domain ontologies metadata from mediators and provides query facilities through OQL. To store

components locally, ComPublish can use the GOA system. To publish and store components on the Web, it uses the *LeSelect* [14] information integration architecture. The mediator interacts with a *LeSelect* client, which acts as a translator, and can access components published and stored by *LeSelect* server publication facilities.

In the following sections we detail the *ComPublish* services starting from the publishing process, then presenting modules and services of the integration layer and *ComPublish* main interface.

3.1 – Publishing sites

To publish and access a component in a remote site through *ComPublish*, the component owner must install *Le Select* system [14]. *Le Select* provides facilities for data and program publishing on the Web. It also provides facilities for publishing metadata associated to the published data.

Based on the work of Guerrieri [15], which uses XML for documenting the various phases of software development, *ComPublish* generates XML documents for describing software components. Thus, for all components to be published, the publisher has to associate an XML description document. In order to provide a uniform component documentation process, pre-defined XML DTDs are created according to the different application domains and component categories. In general, the component description attributes include: the application domain, the application development phase (e.g., analysis, design or implementation), category (e.g., code, diagram), language, and author, among others.

Both the component archive and its related XML document must be informed to *LeSelect* by editing (through a conventional text editor) a *wrapper definition file*, which is a configuration archive (also in XML format) that stores the list of all published data. Finally, the publisher emails the *ComPublish* administrator informing his site identification.

3.2 – Mediation Layer

The main idea behind *ComPublish* is to adapt Heterogeneous and Distributed Data Base System (HDDS) technologies, such as mediators [16] combined with ontology [17, 18] to integrate, identify and retrieve software component repositories instead of legacy databases. Mediators represent and integrate domain information repositories (distributed and/or heterogeneous). Metadata found in mediators describe the repositories of components, presenting the domain, their semantics, software architecture and interfaces. *ComPublish* works with a query engine from the GOA system and therefore

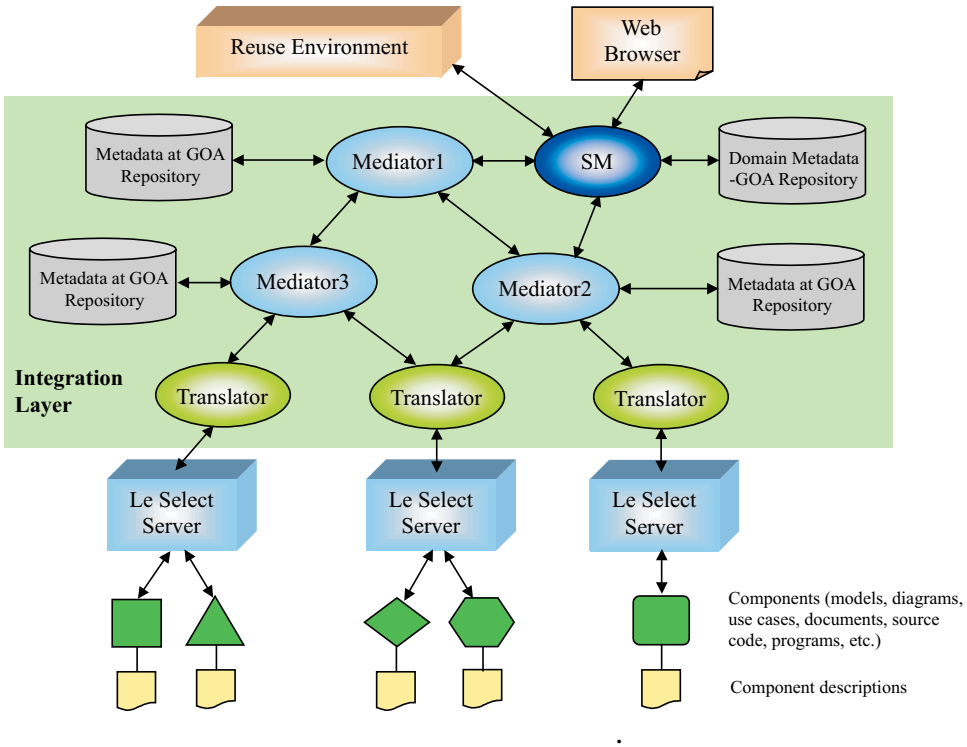

Figure 1: *ComPublish Architecture.*

ad hoc queries on this metadata are used to analyze the available components. The organization of mediators with ontology drives the user search along heterogeneous vocabulary.

ComPublish offers an integrated view of published components at the Internet. *ComPublish* integration layer, based on mediators and wrappers [16], provides the binding of different components to their domain concepts. To assist the identification of related components and their appropriate domain organization, each mediator represents a domain ontology and provides the mapping to their respective components repository. Domain ontologies are used to help the search for reusable components through the representation of domain semantic concepts [10]. Therefore, this mediation layer promotes domain integration and mechanisms to translate component requests across ontologies.

The main services of the mediation layer are:

- Mediators – organizes the mappings between metadata and the data providers associated to the mediator. This metadata includes component descriptions imported from each *Le Select* published site and a hierarchy of ontological terms, both related to the domain represented by the mediator. Each mediator can be activated from different machines interconnected in a network and managed in an independent manner. However, related mediators within an ontology can communicate among themselves by the CORBA protocol.
- GOA Metadata Repository – the component description is transferred via translator to the component application domain corresponding mediator, then integrated to the mediator metadata and stored by GOA object storage system. GOA provides services for storing data as object, and query facilities through OQL. All mediator metadata are stored in a GOA repository. GOA is also responsible for all query processing and optimization services provided by the mediator.
- Translators – are responsible for linking mediators to remote component repositories. Each translator encapsulates a *Le Select* API Client that is able to establish a socket connection to a remote *Le Select* server and ask for its services. These services include metadata import and component download.

3.3 – Service Manager

The Service Manager (SM) is responsible for metadata of mediators, translators and data sources availability, and for dealing with ontological commitments between related mediators (domains). An external application (e.g., a browser or a reuse environment) is able to access the services of *ComPublish* through SM, which can be seen as a special mediator that manages and provides access to all other mediators of the integration layer. SM stores metadata about available mediators, and is capable of creating ontological bindings between related ontologies in order to query several mediators. It is also responsible for the creation and modification of mediators.

4 Searching for Components

The search for components in our architecture, named *CompAgent* combines searching, classification and filtering of components published on the Web, as well as components available from *ComPublish*. *CompAgent* is based on advances on intelligent information retrieval on the Web. Techniques such as user models (profiles), collaborative modeling and recommendations are used for composing the retrieval system.

Figure 2 shows a general overview of *CompAgent*. The main service of *CompAgent* is a meta search, using a Web search engine (i.e., Google Search Engine[1] [19]), and *ComPublish* component searching services. *CompAgent* filters, classifies and merges information coming from these search engines through the following modules (Figure 2):

- **Search Agent** – it is the main element of the *CompAgent* search architecture. The Search Agent (SA) interacts with the Web search engine and the *ComPublish* system, as detailed in section 4.1.
- **Machine Learning Module –** *Machine Learning* techniques are used to observe and learn the behavior of the user while he navigates through domain information or chooses information among suggestions done by the WWW agent. The Feedback Agent, internal to the machine learning module, processes the observed behavior. Another responsibility is to update the recommendation base and collaborative base, which supply information for the algorithm of the filtering agent. This module also provides information that is used to adapt the user profile.

[1] Google was chosen as the search engine for the architecture as detailed in [20]. When compared to AltaVista [21], Google presented many advantages, such as being able to presenting of a more complete Internet index, and a powerful algorithm for classifying the page relevance. Since it is an academic tool, it is possible to use it remotely without any commercial restriction.

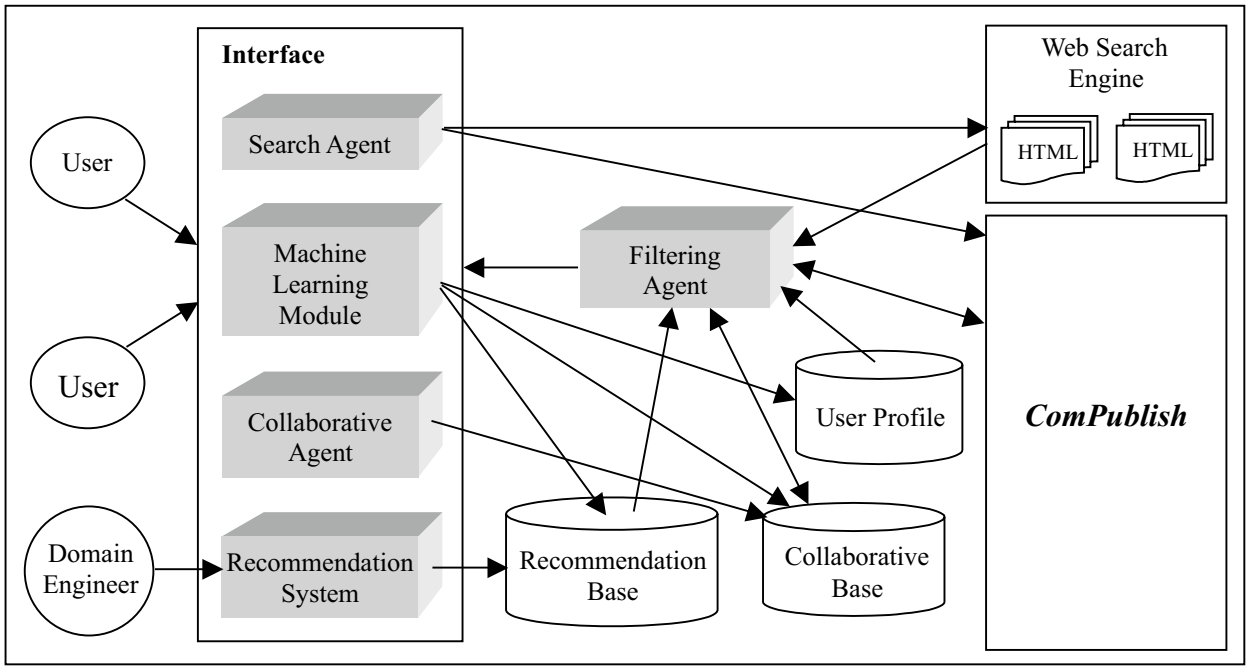

Figure 2: *Componentes Seach Architecture*

- **Collaborative Agent** – this agent interacts with the filtering agent to recommend components based on the existing information about the user stereotype. All users of the same stereotype share links that are considered to be important by one user category. These links are stored on a base, called the collaborative base.
- **Recommendation System** – is responsible for inserting components and information provided by the person in charge of modeling the domain, in this case, the domain engineer. As shown in Figure 3, the degree of relevance of that component and some domain description is informed to the recommendation system. The filtering algorithm uses the recommendations, which are stored as objects in the recommendation base.
- **Filtering Agent –** is responsible for the filtering and organization of component search results. The information of components is presented to the user ranked by the degree of importance.

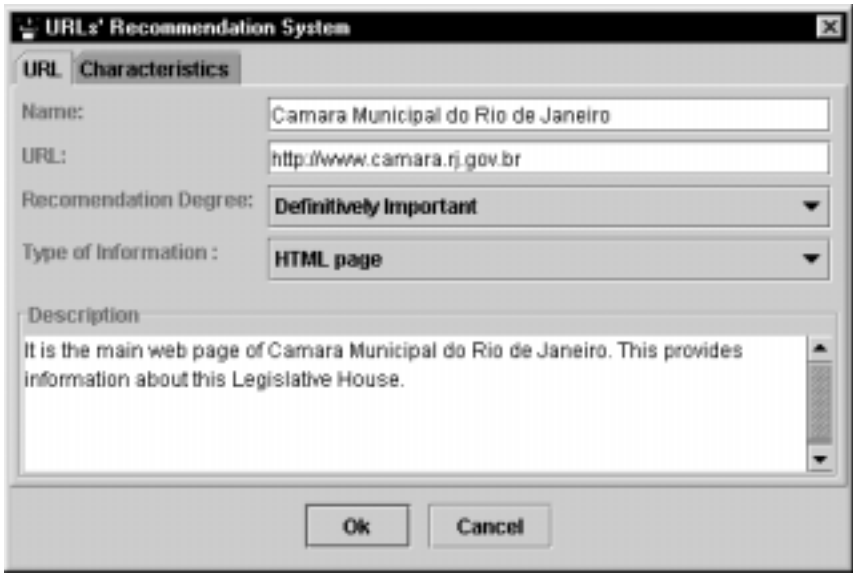

Figure 3: *Recommendation System Manager*

4.1 – Using *CompAgent* Services

First, the user informs (Figure 4) the main characteristic of the component being searched. The filtering degree can be one of the following relevance degrees: Definitively Important, Very Important, Important, Quite Important, and Not Very Important. This term will work as the cutoff value for the returned links by the Web Search Engine and ComPublish.

The next step consists in checking whether that link *i* is already part of the recommendation base or the collaborative filtering base. If the link is found, it means that this link has been previously chosen by a user of the same stereotype, or has been recommended by the domain engineer responsible for the current domain. Therefore, RV_i (Eq. 2) incorporates the previous value (Eq. 1) plus doubling the weights of the links on the collaborative base (w_{rb}) and on the collaborative filtering (w_{cf}) multiplied by the relevance assigned to the link in the specific base. The relevance is represented as r_{rb} for recommendation base and r_{cf} for collaborative filtering.

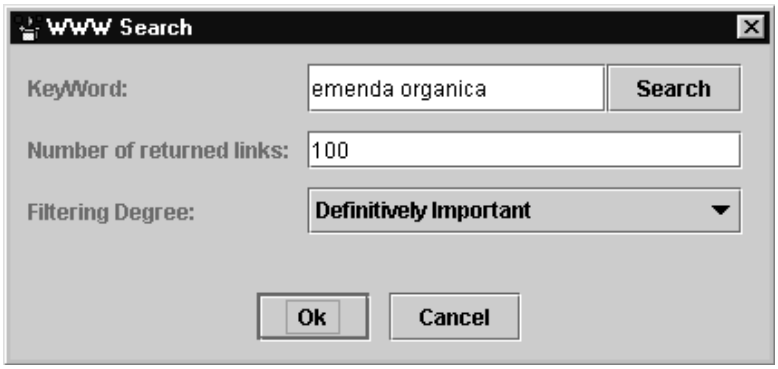

Figure 4: *Search Frame*

The SA composes a message containing the query parameters to be processed internally by *ComPublish*. This message contains the application domain so that *ComPublish* can use the adequate mediators and all attributes the user wants to find. *ComPublish* system receives the message and builds the corresponding query. This query scans the local and remote repositories that are linked to the required domain. Components metadata are searched to match the component features specified by the user. Successful components are listed back to SA in a XML document. This XML document contains all component attributes that were requested by the query. This document is parsed and component attributes are extracted. These attributes (for instance, name, category, date, author, development phase, language, description, among others) are passed to the filtering agent.

The filtering process starts by obtaining a classification measure, RV, in Equation 1, for each of the *n* returned links. Each RV_i value is calculated after function ó that matches the keywords from the user profile against the attribute values extracted from link *i*. In Eq. 1, for each attribute a_a, the number of occurrences of each keyword k_k are counted and multiplied by the degree of relevance (w_k) of the keyword k_k., where *m* is the number of attributes in link *i* and *p* is the number of keywords from the user profile.

$$RV_i \quad \sum_{a\ 1}^{m} \sum_{k\ 1}^{p} (k_k, a_a) \quad w_k \qquad \text{(Eq. 1)}$$

$$RV_i \quad RV_i \quad 2 \quad ((w_{rb} * r_{rb}) \quad (w_{cf} * r_{cf})) \qquad \text{(Eq. 2)}$$

The third step of the Filtering Agent is to consider the ranking of the link returned by the resulting list from Google, mainly due to the reliability degree supplied by the algorithm of PageRank, used internally by this search engine. Equation 3 calculates the new RV_i value, by incorporating the previous RV_i (Eq. 2) and adding a value representing the link rank. This measurement is the difference between the number of returned links (*nl*) and the position of link *i* (pl_i). This value is multiplied by 0.5.

$$RV_i \quad RV_i \quad ((nl \quad pl_i) \quad 0.5) \qquad \text{(Eq. 3)}$$

After sorting the classification value of all links, the one with the highest ranking is taken and serves as the value for filtering the remaining links. The process consists in dividing the classification value of link *i* (RV_i) by the value of the most relevant link (RV_h) (Equation 4). The obtained value is compared to the cutoff value previously informed by the user (Figure 5). In case the division value (DV_i) is smaller than the cutoff value, the link is ignored, and is not inserted in the result to be shown to the user.

$$DV_i \quad RV_i / RV_h \qquad \text{(Eq. 4)}$$

A similar process is applied for links returned by *ComPublish*, but the position of the returned component, as presented in Equation 3, is not considered, since *ComPublish* does not incorporate a classification algorithm for establishing the importance of the link.

Finally, the Web post-processed results are merged and presented to the user (Figure 5). The user can pick the returned link or component that he finds most relevant for his domain engineering process. In case the choice is a component, download is done for the machine of the user. If it is a page, the correspondent page is opened on the default browser of the machine.

Figure 5: *Results Frame*

When the user chooses a link, the feedback process is activated. Initially, the recommendation base is analyzed. If any domain engineer has recommended this link, its weight will be increased by one unit. If the user who is choosing the link is a domain engineer, the link is added to the recommendation base. A similar process is done for links in the collaborative base, but with a small difference, if the chosen link does not exist in the collaborative base of the current user, it is inserted into the base. If the link is already there, its weight is increased by one unit.

5 Using Mediation Services in the Legislative Domain

We have experimented the mediation layer with local components repositories for the legislative domain as part of a project conducted at the Municipal Legislative House of Representatives from Rio de Janeiro (CMRJ). The project aimed at integrating the effort in software development in the legislative domain. Our example considers the development of an application that revises and prepares new proposals for the municipal Code in the legislative domain.

In this application (Figure 6), data source 1 provides a Java package (set of related classes) named “Proposal Creation” and data source 2 has a binary software component called “New Subject”. The Legislative Domain Mediator provides an ontology term named “proposal”, which is associated to the metadata terms “New Proposal” and “New Project” which are mapped to both component data sources. However, there was a previous Judiciary Domain Mediator registered in the architecture. The judiciary domain has an ontology term named “code” that is mapped to a component named “Search Code Database”. Since the proposal creation may involve activities related to pre-existing municipal codes, the SM administrator associated the legislative with the judiciary domain, through a hyponym[2] relationship between the two domain ontologies.

Thus, when our user accesses the ComPublish interface to retrieve components related to the creation of new proposals, he can choose to access information from all related mediators, i.e., generic mediators, specific mediators, associated mediators or all of them. Suppose our user

[2] A hyponym is a type of ontological relationship between two ontological terms related to two different domains [9].

decided to retrieve information from the Legislative Domain and its associated mediators. He would access components from the Legislative and the Judiciary Mediator (see Figure 6). The formulation of the query is based on selecting the component type. For each component, a description is presented and the user can select one or more components to be retrieved.

Through the mediation structure, users can search for components in a transparent and uniform way [10]. In the above example, users do not have to know where components are stored. Moreover, users do not have to query all component repositories, using each repository query language format (when a query language exists) to find where needed components are stored. They do not have do know either how to access data sources.

message containing the query parameters to be processed internally by *ComPublish*. SA is also responsible for opening a connection and sending the query to the Web search engine (i.e., Google). Once the connection is opened, SA controls the number of links to be received, performs the parsing of each received page, extracting information such as title, link, description, description from Google (registered by the user responsible for that page), and passes this information to the Filtering Agent (Figure 5).

6 Conclusions

In this work we address interoperability issues between repositories of software components on the Web. A mediation layer was built on top of the *LeSelect* system, organizing the description of components according to its applica-

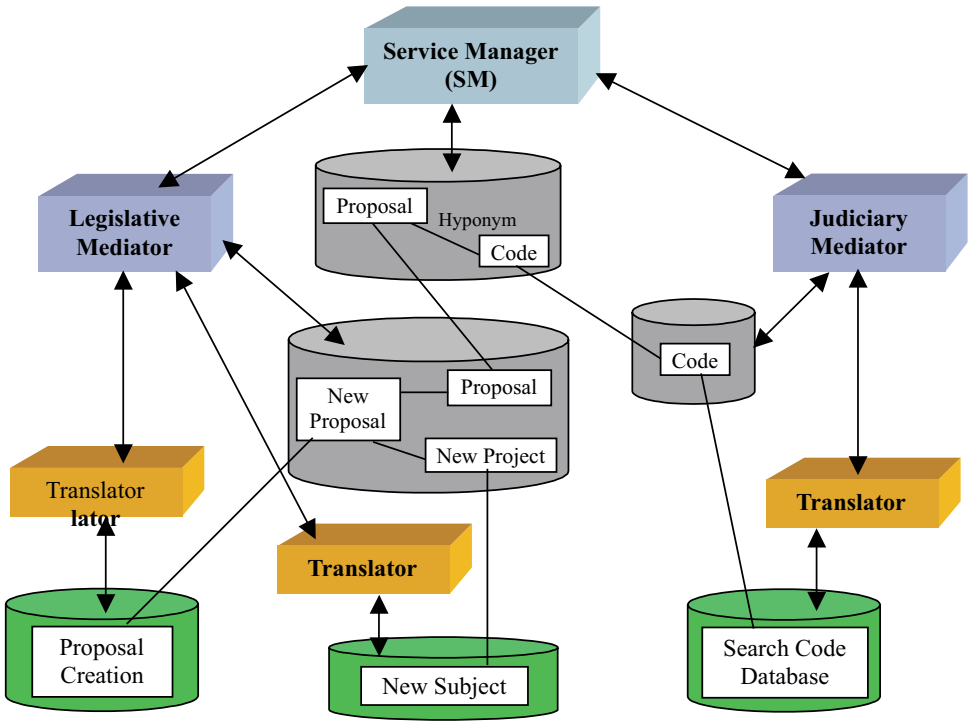

Figure 6: *Mediator Services in the Legislative Domain*

In addition, the user can search the Web in order to find other components related to these domains. In this case, the user may use the *CompAgent*. Suppose the user wants to find components related to the legislative ontological term “emenda orgânica” on the Web. The Search Agent (SA) from *CompAgent* interacts with the Web search engine and with the *ComPublish* system. SA composes a

tion domain. In our architecture, components are described through XML and published by *LeSelect* servers. OQL queries can be issued to the mediation layer and are processed by the GOA object server, which can present query results as a list of suggested components along with its repository link, also in XML.

Without our search engine, if a user has to search for

domain information using the available techniques, such as Web filters, he has to use general Internet search mechanisms such as keyword-based spiders. In this kind of search, the user is probably presented with a lot of irrelevant information. Besides, even if he finds some interesting site, the available domain information might not be in the adequate format, requiring some kind of conversion. Moreover, there are no agents that guide the user to more interesting related information, as we do in our work.

We believe that the component search and publication mechanism provided by our architecture can improve software development based on component reuse. Our approach allows users to express component requests at a higher level of abstraction when compared to keyword based access or component interface based access. The innovative aspect of our proposal is the use of domain engineering with mediators, for reusable component retrieval, both on the Web and on registered repositories. Currently, there is an operational prototype, implemented in Java and C++, with filtering agents and a mediation layer with local repositories. The coupling with *LeSelect* architecture is, actually, running and provides remote publishing of components.

References

[1] Jacobson, I.; Griss, M.; Jonsson, P.; "Software Reuse: Architecture, Process and Organization for Business Success"; Addison Wesley Longman, May, 1997.

[2] Seacord, R.; Hissan, S.; Wallnau, K, "Agora: A Search Engine for Software Components", IEEE Internet Computing, vol.2, no.6, November/December,1998, pp. 62-70.

[3] *ComponentSource*, www.componentsource.com. Accessed in 9/Apr/2003.

[4] Sprott, David: "Software Components Marketplace Reality". Interact Journal, A CBDi Forum Publication in www.cbdiforum.com, May 2000.

[5] RIG; "Reusable Library Interoperability Group" at http://www.asset.com/rig/, 1996.

[6] Prieto-Díaz, R.; Implementing Faceted Classification for Software Reuse; Communications of the ACM, vol.34, no.5, May 1991.

[7] Ye, Y.; Fischer, G.: "Promoting Reuse with Active Reuse Repository Systems", IEEE ICSR 2000, Vienna, June 2000, pp. 302-317.

[8] Werner, C.; Braga, R.; Mattoso, M. "Odyssey: A Reuse Environment based on Domain Models"; Proceedings of IEEE Symposium on Application-Specific Systems and Software Engineering Technology (ASSET'99), Texas, 1999, pp. 49-57.

[9] Braga, R.; Costa, M.; Werner, C.; Mattoso, M. "A Multi-Agent System for Domain Information Discovery and Filtering", XIV Brazilian Symposium on Software Engineering, João Pessoa, October 2000, pp.179-194.

[10] Braga, R., Werner, C.; Mattoso, M.: "The Use of Mediation and Ontology Technologies for Software Component Information Retrieval", Proceedings of ACM Symposium on Software Reusability (SSR'01), Toronto, May 2001, pp.19-28

[11] Braga, R.; Mattoso, M.; Werner, C.: "Using Ontologies for Domain Information Retrieval", in DEXA 2000 DomE Workshop, September 2000, pp. 100-104.

[12] Pinheiro, R.: "ComPublish: A System for the Publication, Search and Retrieval of Software Components on The Internet". Master´s Thesis, PESC/COPPE/UFRJ, 2002 (in Portuguese).

[13] Mattoso, M. et al. "Persistency of components in a reuse environment", XIV Brazilian Symposium on Software Engineering, João Pessoa, October 2000, pp.251-254 (in Portuguese).

[14] *LeSelect* - A Mediator System Developed at the Caravel Project. INRIA, France, In http://www-caravel.inria.fr/LeSelect/, Accessed in 9/Apr/2003

[15] Guerrieri, E.: *"Software Document Reuse with XML"*, Proceedings of ICSR-5, Victoria, BC, Canada, June, 1998, pp. 246-254.

[16] Wiederhold, G.: "Mediators in the Architecture of Future Information Systems"; *IEEE Computer Society Press*, Vol.25, March 1992, pp. 38-49.

[17] Wiederhold, G.; Jannink, J.: "Composing Diverse Ontologies"; 8th Working Conference on Database Semantics (DS-8), Rotorua, New Zealand, January 1999.

[18] Nieto, E. M.: OBSERVER: An Approach for Query Processing in Global Information Systems based on Interoperation across Pre-existing Ontologies, Doctoral Thesis, Universidade de Zaragoza, November 1998.

[19] Google Search Engine, http://www.google.com, Accessed in 9/Apr/2003.

[20] Costa, M.: "CompAgent: A tool for support Domain oriented information search and Retrieval on the Web". Master´s , PESC/COPPE/UFRJ, 2002 (in Portuguese).

[21] AltaVista Search Engine, http://www.altavista.com, Accessed in 9/Apr/2003.