Component Based Development (CBD) aims at constructing software through the inter-relationship between preexisting components. The main goal of this work is to provide access to components that can be 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 registered known components from a virtual library of components. In our architecture, components are described through XML and published by local repositories or remote servers. The innovative aspect of our proposal is the use of domain ontologies, for reusable component retrieval. OQL queries can be issued to the mediation layer and are processed by the GOA object server, which can present the query results as a list of suggested components along with its repository link, also in XML.

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 this work 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.

Many potentially reusable components are available on the web. Finding an adequate component involves searching among heterogeneous descriptions of the components within a broad search space. Basically there are two approaches for components search: (i) web search based on components interface; and (ii) a library of components that provides a semantic description of the components and the storage of the component. Most works concerned with finding adequate preexisting components adopt basically either the web approach [2] or the library approach [3, 4].
The problem with the web approach is that components are published and described in many heterogeneous ways and a keyword based search can be very generic. The library approach does not have these disadvantages but is restricted to previously and locally stored components. We present an architecture that combines the web with the library approach. By using a mediation layer we integrate the semantics of web components with registered known components from a virtual library of components. The definition of the semantics of mediators is based on information from a specific application domain. This architecture provides three main layers: (i) published components; (ii) mediation layer with component semantics and ontology services; (iii) multi-agent layer that searches and filters retrieved components from the web as well as from previously registered published components.

These services are part of a software reuse environment, named, Odyssey [5], that aims at providing support in the development and reuse of components in all phases of software construction. The solutions we present here are an extension of the Odyssey search engine [6] to address the publication of components on the Internet. Our reuse environment is motivated by the InterLegis project that is being conducted in the Legislative House of Representatives from the Rio de Janeiro city. There are several applications that can benefit from reusable information within the legislative and related domains, such as judiciary and criminal domain. 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 with related works from the literature. The publication of components with mediation services is presented in Section 3 illustrated by the InterLegis project. The several agents that filter and present potential components for software reuse are discussed in Section 4 and Section 5 presents our concluding remarks.

2. Related Works

The work presented in [2] describes a search engine for the retrieval of reusable code components, such as JavaBeans and CORBA components. The Agora system uses an introspection mechanism for registering code components, through its interface. The Agora System is a web based search approach that searches only on component interfaces, covering solely the component connectiveness problem.

Another important work is the RIG initiative [3], which describes a reuse library interoperability approach. The idea of the asset libraries’ interoperability is based on the storage of domain information in several databases. These databases are static and based on a unique global model.

The Agora system only deals with code components whereas our approach deals with domain information in all abstraction levels, including code components. Moreover, new information is always associated to domain terms within a given domain ontology, improving its accessibility and reuse. 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.

3. Publication of Components

ComPublish is an architecture that aims at publishing software components, 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: (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 to retrieve software components.

Figure 1 shows the architecture of ComPublish services. The integration of component descriptions is mapped on mediators. The Object Server GOA [7] stores 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, we are working on using the LeSelect [8] information integration architecture. The mediator interacts with a LeSelect client that can access components published and stored by LeSelect server publication facilities.

3.1 – Mediation Layer

This layer is based on mediators and ontologies to provide 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.

Mediators provide a uniform view of the available components organized in a domain taxonomy. Domain ontologies are used to help the search for reusable components through the representation of domain semantic concepts [9]. Therefore, this mediation layer promotes domain integration and mechanisms to translate component requests across ontologies.

The main services of the mediation layer are the Service Manager (SM), Metadata Manager (MM) and the Query Manager (QM). The SM stores metadata about available mediators, and is capable of creating ontological bindings between related ontologies in order to query several mediators. Also, the SM is responsible for the creation and modification of mediators. The MM organizes the mappings between metadata and the data providers associated to the mediator. These metadata include a hierarchy of ontological terms related to the domain represented by the mediator. A relationship between domain ontological terms and components is established and stored in the GOA metadata repository. However, the relationships between different ontologies are kept at the Service Manager. Access to

![Figure 1 – ComPublish architecture](image)
mediators is done through the GOA QM, which process OQL queries on the semantics of the components. Query results on component descriptions can be delivered in XML format.

3.2 – Publishing sites

The publication of a component requires the installation of a LeSelect server on the publishing site. Components are described through XML using metadata services of the LeSelect server. A predefined XML DTD is provided to help in having a uniform documentation of components. A component description is based on domain information. The description attributes to register the required components are: the application domain, the phase of application development of the component (i.e. analysis, architectural or implementation), type (i.e. code, diagram), language, author, among others, such as:

```
<component>
    <domain> Legislative </domain>
    <phase> analysis </phase>
    <type> use case </type>
    <language> UML </language>
    <author> Robson Pinheiro </author>
    ...
</component>
```

3.3 – Using Mediation Services in the Legislative Domain

We are experimenting this mediation layer with local components repositories for the legislative domain as part of the InterLegis project. InterLegis aims 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 2), data source 1 has 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 has an ontology term named “proposal”, which is associated to the metadata terms “New Proposal” and “New Project” that is 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 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 decided to retrieve information from the Legislative Mediator and its associated mediators, then he would access components from the Legislative and the Judiciary Mediator (see Figure 2). 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 [9]. 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.

4. Searching for Components

The search for components in our architecture combines searching and filtering components published on the web as well as components available from ComPublish. Searching is provided by the multi-agent architecture, presented in Figure 3. The multi-agent system is based on advances on discovery and filtering of information at the Internet. Techniques such as user models (profiles), and recommendations were used for composing a navigation multi-agent system.

The main multi-agent services are:

- **Search Agent** – is responsible for mapping queries to the Web Search Engine and the ComPublish architecture;
- **Machine Learning Module** – Machine Learning techniques are used to observe and learn the user behavior while the user navigates through domain information. Also, it is possible to infer weights based on the user interests, by considering the number of occurrences of words in documents that are frequently seen by the user, or patterns of navigation paths repeatedly
followed by him. This module provides information to build the user profile and to rank links. The user profile also provides stereotypes (users that share similar profiles);

- **Collaborative agent** – this agent interacts with the filtering agent to recommend components based on the information provided by the machine learning module;

- **Filtering Agent** – is responsible for the filtering and organization of component search results. The information of components is presented to the user ranked by a degree of importance;

The user provides, through the interface, the main feature of the component being searched for. The Search Agent (SA) interacts with the web search engine and the ComPublish system. 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 also informations based on the user profile and main component features.

The ComPublish system receives the message and builds the corresponding query. This query will scan the local and remote repositories that are linked to required domain. Components metadata are searched to match the component features specified by the user. Successfull components are listed to the Filtering Agent, containing its name, description and repository link.

The Search Agent is also responsible for sending the query to the web search engine, which is currently based on Google [10] results. These results are processed on the Filtering Agent, which improves the ranking based on user profile and hot links. Finally resulting links from ComPublish and web post-processed results are combined and presented to the user.

5. Conclusions

In this work aims 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 application 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 the 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 components 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 ontologies, 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 providing remote publishing of components is under development.

References


