Software Engineering Component Repositories

Robert C. Seacord

Software Engineering Institute

Carnegie Mellon University

Pittsburgh, PA 15213 USA

+1 412 268 7608

rcs@sei.cmu.edu

ABSTRACT

Traditional, large-scale software repositories have historically failed, principally as a result of their conception as centralized systems. New and emerging technologies such as traders, brokers, location services and search engines have yet to be proven effective in the location and adoption of reusable software components. The Component-Based Systems (CBS) Initiative at the Software Engineering Institute (SEI) developed the Agora software prototype to investigate the integration of search technology with component introspection to create a distributed, worldwide component repository. This paper provides a description of Agora, its strengths and shortcomings, and discusses the evolution of component-based software engineering necessary to support an effective component repository.

Keywords

Components, Search Engine, COTS, CBSE, JavaBeans, Software Repository.

Introduction

Agora is a prototype component repository being developed by the Software Engineering Institute at Carnegie Mellon University [1] [2]. The object of this work is to create an automatically generated, indexed, database of software components classified by component model (e.g., JavaBean, ActiveX, CORBA, Enterprise JavaBean). Agora combines introspection with Web search engines to reduce the costs of bringing software components to, and finding components in, the software marketplace.

The benefits of developing an effective component library are readily apparent: by allowing system integrators to fabricate software systems from pre-existing components rather than laboriously develop each system from scratch, enormous time and energy can be saved in the development of new software systems. The President's Information Technology Advisory Committee (PITAC) interim report [3] to the President states that:

The construction and availability of libraries of certifiably robust, specified, modeled and tested software components would greatly aid the development of new software.

However beneficial a component library might be, a useful and effective repository has turned out to be an elusive goal. Traditional software libraries have been conceived as large central databases containing information about components and, often, the components themselves. Examples of such systems include the Center for Computer

Systems Engineering's Defense System Repository, the JavaBeans Directory, and the Gamelan Java directory.

While the JavaBeans and Gamelan directories are still going concerns, similar systems have failed in the past largely as a result of their conception as centralized systems. Problems with this approach include limited accessibility and scalability of the repository, exclusive control over cataloged components, oppressive bureaucracy, and poor economy of scale (few users, low per-user benefits, and high cost of repository mechanisms and operations).

Search engines are a rapidly evolving Web technology that has the potential to solve the conundrum of a useful component library. Existing search engines provide convenient support for different kinds of Web content. Different search capabilities are provided for different types of content. For example, text content can be searched by simple but effective pattern matching, while images can be searched only by image name.

The AltaVista search service, for example, supports special functions for Web searches. In particular, searches of the format: "`applet:class`" can locate HTML pages containing applet tags where the code parameter is equal to specified Java applet class. For example, a search for "`applet:sine`" can be used to find applets where the code parameter is specified as "`sine`" or "`sine.class`". While this approach has obvious advantages of simply searching for the term, it still only allows the name of the components to be indexed and searched.

The Agora search engine enhances existing but rudimentary search capabilities for Java applets. By using Java introspection, the Agora search engine can maintain a more structured and descriptive index that is targeted to the type of content (the component model) and the intended audience (application developers) than is supported by existing search engines. For example, information about component properties, events, and methods can be retrieved from Agora.

Issues, Tradeoffs & Future Directions

Agora was designed and implemented to demonstrate the feasibility of a component repository using existing infrastructures and available information. Agora demonstrates the extent to which an automatically indexed, database of software components can be implemented given the current state of the practice. Agora does not address all the issues that need to be resolved before such an approach can be effectively used on a broad scale. Some of the tradeoffs between the approach taken by Agora and more traditional repositories are discussed in this section. Areas in which component-based software engineering may evolve to more effectively support component repositories are identified.

Modeled

The PITAC report stipulates that components available in a repository must be fully *modeled*. This may mean that a behavioral model of the component has been developed. Alternately, it could mean that the component adheres to a predefined component model.

A component model describes the coordination model used by subscribing components so that they may be seamlessly integrated into a system that applies the model. The most prominent component technologies including Enterprise JavaBeans, and ActiveX all impose constraints on components [4]. Existing CORBA servers do not really meet the requirements of being a component model, but are still of interest as the OMG is planning on adopting a component model as part of the CORBA 3.0 specification.

Agora was designed to support search and retrieval of multiple component models, but only JavaBeans were fully developed. Some experimentation with CORBA was attempted, but results were not promising due to the difficulty in locating and introspecting CORBA servers. A component repository should be able to locate, index and retrieve a broad variety of components.

Traditional repositories often collected software products, language specific subroutines, or link-able libraries. In removing sources of architectural mismatch [5], evolving component models should improve the effectiveness of the software repository concept.

Interface Descriptions

An obvious problem or limitation of Agora is the lack of descriptive information about the component's interface. In the case of JavaBeans, for example, information that can gathered through introspection is principally restricted to method signatures including function names, return and parameter types. In other component models, such as CORBA servers, interface information is maintained externally to the component and may not be available at all. In all cases, descriptive information about the overall purpose of each component and the various APIs is lacking. In addition, information about functional semantics is completely absent.

Traditional repositories are better positioned to provide interface descriptions than Agora, by documenting the interfaces according to some standard format used by the repository or making the original component documentation available within the repository.

The PITAC report stipulates that components available in a repository must be fully specified. Existing component models do not provide a specification that is sufficiently detailed to allow programmers to take advantage of the component without reference to further documentation or excessive experimentation.

JavaBeans, for example, supports introspection to determine the signatures of the class methods but does not provide a description of the semantics of the calls. The semantics of the API calls is normally described, albeit in an informal manner, in documentation for the API. Although this documentation is not available in a JavaBeans' executable form, the information is often available in the source code in the form of structured comments. The `javadoc` tool parses the declarations and documentation comments in a set of Java source files and produces a corresponding set of HTML pages describing (by default) the public and protected classes, inner classes, interfaces, constructors, methods, and fields. This information could also be converted by a doclet into a runtime accessible format.

As demonstrated in WaterBeans [6], component models could also provide a means of including contact information so that the original author(s) may be contacted.

Quality Assurance

The quality of components in traditional software repositories is often assured by the organization that maintains the repository. There are number of problems with this approach:

1. Timeliness – having a single organization responsible for providing quality assurance for every version of very component created is an impossible goal. Taking this approach instantly creates a bottleneck, restricting the number of components that can be incorporated into the amount of time it takes to get a component reviewed and incorporated into the repository.

2. Lack of context – components must be qualified as being able to perform a specific task. A component that is qualified to perform a certain function may not be suitable, for example, for a real time application.

3. Bias – it is highly unlikely that the organization maintaining the component repository is completely unbiased. Most organizations that might be capable of providing this service have their own products, customers, and strategic relationships and partnerships. It would be naïve to assume these relationships will not impact, at the very least, the components that are included in the repository.

The Agora model is based on the premise that component databases need to be free and inclusive. Value-added industries such as consumer reports and underwriter labs can add value by providing independent quality assurance of popular components. The existing Agora prototype would be extended to allow underwriter labs to link product evaluations to specific components maintained within the repository. Again, this process will be handled in a completely decentralized fashion. Potential consumers can review these reports and form their own opinions as to the reputation of the organization providing the information and the value of the report. It may be also possible to automatically generate mailing lists on a per component basis to let consumers of that component directly exchange experiences.

Components in the repository (or elsewhere) can be digitally signed to indicate that the provide specific quality of service attributes, for example that they are guaranteed to execute in a specified period of time (for use in real time systems) or that the component has completed some battery of tests. The objects can be signed directly by the certifying agency. Providing tamper-proof packaging will increase the level of trust of consumers that the component being evaluated does in fact have the certified qualities.

Open System

Traditional component repositories were conceived as centrally managed systems. This allowed the group or organization maintaining the repository to certify the degree to which components in the repository were robust, specified, modeled and tested. Without central management, it is very difficult to ensure the quality of the components in the repository.

In the implementation of the Agora prototype, it was felt that component repositories need to be, at first, free and inclusive. Agora automatically compiles indexes by going out over the Internet and discovering and collecting information about software components. Component collection is performed in a nonjudgmental manner, so the problem of having a sole arbiter decide what does and does not belong in the repository is eliminated.

The traditional, centralized approach is analogous to a centrally planned economy in that both are slow to respond to market changes and often succeed only when the investment outweighs the benefits. The free and inclusive approach can be compared to a market driven economy that is more responsive to market realities but does not provide the safeguards of the more rigidly planned system.

Component Uniqueness

A goal of a component repository is to incorporate each component once and only once, but this is easier said than done. There are a number of different ways to determine if a component is unique. Here are a sample of methods that can be used to determine uniqueness and the associated drawbacks:

1. **The component has a unique URL.** Unfortunately, multiple identical copies of a component often appear at multiple locations so this approach does not guarantee a component only appears once in the database.

2. **The component has the same application programming interface (API).** Since the API can be introspected, it can be compared with other APIs as a test for uniqueness. However, multiple different versions of a component could easily have identical interfaces. Another potential issue is if different versions of the same component should be considered to be different components or not.

3. **The component matches byte for byte with an existing component.** This is a relatively restrictive test for uniqueness although there are still potential problems. For example, an Enterprise JavaBean may be modified at deployment time to support specific characteristics defined in the deployment descriptor. It is therefore possible that the same Enterprise JavaBean deployed in one environment will not match the same Enterprise JavaBean deployed elsewhere.

Component models should have a means of specifying a major and minor version number in the component. This could provide a useful mechanism for differentiating two components that otherwise have the same API.

Data Rights and Privacy

Agora automatically collects components discovered over the Internet using a spider. This raises some interesting questions regarding data rights and privacy.

Spiders follow HTML links in existing documents. Most of these documents have been placed in locations where they are accessible to the public. There are probably some cases where confidential or proprietary documents have inadvertently been made accessible to the public but this is an exceptional occurrence.

Java classes are often included on web pages to increase the dynamic content of the page. In most cases the components are not provided for the express means of collection and integration by other system integrators.

Although the source code used to generate these components can be protected under copyright laws, there is no defined mechanism for protecting the binary copies of a program other than preventing copies from being distributed or integrating some manner of licensing software.

ActiveX controls, for example, can be built to work only at design time, runtime or in either situation using LPK files.

Electronic Commerce in Components

Closely related to the problem of data rights and privacy of components is electronic commerce in components. Most publicly accessible components are freely available, non-proprietary, and non-commercial. However, because of the degree of investment required, many types of components may only be available commercially. However, vendors of commercial components are very unlikely to make these components publicly available unless they have some means of protecting their investments.

There are a number of potential solutions for this problem. One solution would require the development of an electronic commerce model where components could be "rented". An initialization call could, in fact, provide a credit card number that can be electronically authorized (in a similar fashion to any retail store). Subsequent calls to the component could be charged against the credit card. Depending on the expected calling frequency, this might require the use of *nano-bucks* – extremely small measures of currency.

Another solution would be that commercial companies make skeleton versions of components publicly available as a means of advertising the features of the component. The system integrator would then need to contact the vendor to license the component prior to employing it. ActiveX controls, for example, could be built to work only at design time. This would allow the controls to be indexed by an automated component repository while the development organization retained control of component distribution.

It is easy to imagine other schemes that could also be employed to achieve similar results.

Software Engineering

Introspection, as defined by the JavaBeans specification, provides a means for development tools, such as the BeanBox, Borland's Jbuilder™, IBM's Visual Age® for Java and Symantec's Visual Cafe to discover component interfaces at runtime. This allows developers to integrate components within a development environment without having to "teach" the development tool about the component.

This same introspection capability made possible the indexing of interface information by Agora. A Component repository can be thought of as a software engineering tool used by system integrators to develop component-based systems. This is relevant because any enhancement to existing component models to support component repositories will generally benefit the broader class of development tools. For example, extending the JavaBean component model to support the description of the API would allow development tools such as the BeanBox to provide on-line documentation for API calls at run-time to assist the developer in the integration of the component.

Location Services

In general locating components in the Agora model is problematic. In developing Agora, we found a ready supply of JavaBeans that have been used as applets on World Wide Web pages. These applets can then be easily found by normal spidering techniques.

JavaBean components found by this method are typically used to add dynamic behavior to static HTML-based Web pages. While this is a potentially useful area of components to be made available in a component repository, it does not represent the full range of JavaBeans that may be available. While it is expected that this Web-based method could also be used to successively retrieve ActiveX components, it may be less effective in finding CORBA servers, Enterprise JavaBeans, or other component types.

CORBA defines multiple, competing mechanisms for locating CORBA servers including implementation repositories, location services, naming services and Object Trader Services. Implementation repositories and location services are not generally useful outside of specific subnet. Object Trader Services require that components be registered, a process that often requires fees. Locating CORBA objects in a naming service turned out to be problematic for several reasons. First, the majority of CORBA servers do not store their object references in a naming service. Second, even if they did, there is no good bootstrapping process for finding an initial object reference for the naming service. This problem could be addressed by having naming services respond to queries on a well-known standard port number or providing some sort of meta-naming service. The best opportunity to discover a naming service is to look for them at vendor-supplied default port numbers.

Agora currently provides a means for component developers to register their components at the Agora web site. This allows components that cannot be located using existing location techniques to be included in the repository.

Summary and Conclusions

The issues associated with developing a useful and effective software repository are distributed across a range of technology areas.

Existing component models need to be extended so that additional information about the component can be accessed at runtime, particularly a description of the semantics of the API calls and the purpose of the component.

Existing naming and directory services need to be standardized, so that automated search tools can search well known port numbers to find and access registered components.

Component developers need to take advantage of existing capabilities (such as the naming service in CORBA) and integrate enhancements to these component models as they become available.

It is unlikely that these varied technology areas will all converge on useful solution to the software repository problem without education and direction from a central group advocating the establishment of these software engineering repositories.

AcknowledgEments

Neil Christopher at the National Institute of Standards and Technology Manufacturing Engineering Lab (NIST/MEL) sponsored this work. Special thanks to Kurt Wallnau and Scott Hissam who helped develop the ideas and the Agora prototype and reveiwers Dan Plakosh, Chuck Buhman and John Foreman.

REFERENCES

1. Seacord, R., Hissam, S., and Wallnau, K., "Agora: A Search Engine for Software Components." IEEE Internet Computing, 2: 6, November/December, 1998, pgs. 62-70.

2. Seacord, R., Hissam, S., Wallnau, C., Agora: A Search Engine for Software Components, CMU/SEI-98-TR-011, August 1998.

3. President’s Information Technology Advisory Committee Interim Report to the President, National Coordination Center for Computing, Information, and Communications, Arlington, VA, August 1998.

4. Alan Brown, Kurt Wallnau, The Current State of Component-Based, Software Engineering (CBSE) IEEE Software, September 1998, pg. 37-47.

5. Garlan, D., Allen, R., and Ockerbloom, J., Architectural Mismatch: Why reuse is so hard. IEEE Software, November 1995.

6. Plakosh, D., Smith, D., Wallnau, K., Building a Custom Component Model: Concepts and Application to Water Quality Models, to be published as an SEI Technical Report, Spring 1999.