IOSR Journal of Computer Engineering (IOSR-JCE)
e-ISSN: 2278-0661, p- ISSN: 2278-8727Volume 16, Issue 3, Ver. I (May-Jun. 2014), PP 100-105
www.iosrjournals.org

Semantic Based Amalgamation of Pulverized Components Using Ontology

Mr. Susheel George Joseph M.C.A, M.Tech, M.Phil(CS)
(Assistant Professor, Department of M.C.A, Kristu Jyoti College of Management and Technology, Changanassery, Kerala. susheelgj@gmail.com)

Abstract: *The problem faced by both software developers and user in development environment and user environment is finding compatible and complementary components in large set of different applications. Different application built in different environment, at different time, by different technology creates a chaotic situation for user and developer to select a required component from vast variety of applications. The problem can be solved by applying technologies related to semantic web. Ontology is created which relates pulverized components based on requirements.*

Keyword: *Semantic Web, Ontology, Pulverization and amalgamation.*

I. Introduction

The Semantic Web provides a common framework that allows data to be shared and reused across application, enterprise, and community boundaries. It is a collaborative effort led by W3C with participation from a large number of researchers and industrial partners. It is based on the Resource Description Framework (RDF), which integrates a variety of applications using XML for syntax and URIs for naming, a concept proposed by World Wide Web's inventor Tim Berners-Lee. The Semantic Web is based on Tim Berners-Lee's World Wide Web idea, that the Web as a whole can be made more intelligent and perhaps even intuitive about how to serve a user's need. Berners-Lee observes that although the search engine indexes have much of the Web's content, they have little ability to select the pages that a user really want or need. He foresees a number of ways in which developers and authors can use self-descriptions and other techniques so that context-understanding programs can selectively find what users want.

The Semantic Web is focused on machines. The Web requires a human operator, using computer systems to perform the tasks required to find, search and aggregate its information. It's impossible for a computer to do these tasks without human guidance because Web pages are specifically designed for human readers. The Semantic Web is a project that aims to change that by presenting Web page data in such a way that it is understood by computers, enabling machines to do the searching, aggregating and combining of the Web's information — without a human operator.

Tim Berners-Lee originally expressed the vision of the semantic web as follows

I have a dream for the web [in which computers] become capable of analyzing all the data on the web – the content, and transaction between people and computers. A 'Semantic Web', which would make this possible, has yet to emerge, but when it does, the day- to –day mechanism of trade, bureaucracy and our daily lives will be handled by machines talking to machines. The 'Intelligence agents' people have touted for ages will finally materialize.

Tim Berners-Lee,1999

The Semantic Web is not about links between web pages. It describes the relationships between things (like A is a part of B and Y is a member of Z) and the properties of things (like size, weight, age, and price). The Semantic Web is not a separate entity from the World Wide Web. It is an extension to the Web that adds new data and metadata to existing Web documents, extending those documents into data. This extension of Web documents to data is what will enable the Web to be processed automatically by machines and also manually by humans. To do this RDF (**R**esource **D**escription **F**ramework) is used to turn basic Web data into structured data that software can make use of RDF works on Web pages and also inside applications and databases.

II. Semantic Web Architecture

The first layer, URI and Unicode, follows the important features of the existing WWW. Unicode is a standard of encoding international character sets and it allows that all human languages can be used (written and read) on the web using one standardized form. Uniform Resource Identifier (URI) is a string of a standardized form that allows to uniquely identifying resources (e.g., documents). A subset of URI is Uniform Resource Locator (URL), which contains access mechanism and a (network) location of a document - such as http://www.example.org/. Another subset of URI is URN that allows identifying a resource without implying its location and means of dereferencing it. The usage of URI is important for a distributed internet system as it provides understandable identification of all resources. An international variant to URI is Internationalized Resource Identifier (IRI) that allows usage of Unicode characters in identifier and for which a mapping to URI is defined.

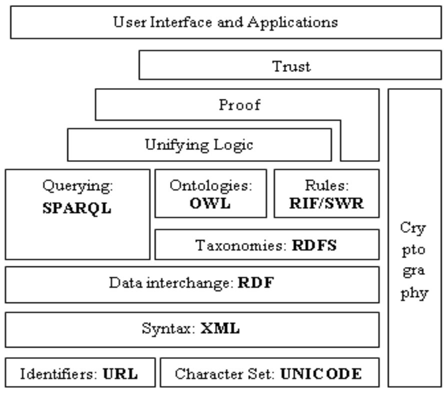

Fig. 1 Semantic web architecture in layers

Extensible Markup Language (XML) layer with XML namespace and XML schema definitions makes sure that there is a common syntax used in the semantic web. XML is a general purpose markup language for documents containing structured information. A XML document contains elements that can be nested and that may have attributes and content. XML namespaces allows specifying different markup vocabularies in one XML document. XML schema serves for expressing schema of a particular set of XML documents. A core data representation format for semantic web is Resource Description Framework (RDF). RDF is a framework for representing information about resources in a graph form. It was primarily intended for representing metadata about WWW resources, such as the title, author, and modification date of a Web page, but it can be used for storing any other data. It is based on triples subject-predicate-object that form graph of data. All data in the semantic web use RDF as the primary representation language. The normative syntax for serializing RDF is XML in the RDF/XML form. Formal semantics of RDF is defined as well. RDF itself serves as a description of a graph formed by triples. Anyone can define vocabulary of terms used for more detailed description. To allow standardized description of taxonomies and other ontological constructs, a RDF Schema (RDFS) was created together with its formal semantics within RDF. RDFS can be used to describe taxonomies of classes and properties and use them to create lightweight ontologies.

More detailed ontologies can be created with Web Ontology Language OWL. The OWL is a language derived from description logics, and offers more constructs over RDFS. It is syntactically embedded into RDF, so like RDFS, it provides additional standardized vocabulary. OWL comes in three species - OWL Lite for taxonomies and simple constrains, OWL DL for full description logic support, and OWL Full for maximum expressiveness and syntactic freedom of RDF. Since OWL is based on description logic, it is not surprising that a formal semantics is defined for this language. RDFS and OWL have semantics defined and this semantics can be used for reasoning within ontologies and knowledge bases described using these languages. To provide rules beyond the constructs available from these languages, rule languages are being standardized for the semantic web as well. Two standards are emerging - RIF and SWRL. For querying RDF data as well as RDFS and OWL ontologies with knowledge bases, a Simple Protocol and RDF Query Language (SPARQL) is available. SPARQL is SQL-like language, but uses RDF triples and resources for both matching part of the query and for returning results of the query. Since both RDFS and OWL are built on RDF, SPARQL can be used for querying ontologies and knowledge bases directly as well. SPARQL is not only query language; it is also a protocol for accessing RDF data. It is expected that all the semantics and rules will be executed at the layers below Proof and the result will be used to prove deductions. Formal proof together with trusted inputs for the proof will mean that the results can be trusted, which is shown in the top layer of the figure: 1. For reliable inputs, cryptography means are to be used, such as digital signatures for verification of the origin of the sources. On top of these layers, application with user interface can be built.

Need for the System

The existing techniques, technologies, and algorithms used for finding and matching Web service applications (WSDL-based) which can be reused, with only insignificant changes, for the purpose of finding well-suited and complementary non-Web-service-based applications for amalgamated applications. This mechanism may include graphical user interfaces, which are not artifacts described in Web service applications.

By building on the techniques initially developed for Web services matching, finding useful and valid applications for amalgamated applications using high-level concepts is possible. This enables the progressive construction of amalgamated applications by end users from a catalog of available applications without deep knowledge of the language. This is an advantage over existing pulverize up tools which require mastering of varying degrees of encoding skills.

Objective of the System

An application of semantic annotation together with the standard Semantic Web matching algorithm for finding sets of functionally equivalent components out of a large set of available Non-Web-service-based components. Once such a set is identified, the user can drag and drop the most suitable component into an Eclipse-based composition canvas. After a set of components has been selected in such a way, they can be connected by data-flow arcs, thus forming an integrated, composite application without any low-level programming and integration efforts. The unique characteristics of a component (i.e., coexistence of graphical user interface description) and new techniques of merging semantic descriptions across multiple components, a much more accurate search result for compatible components can be achieved. As an example: CityStatePicker is implemented as an Eclipse UI Extension class, which allows user to select a state and a city from two separate drop down lists. After both city and state have been selected, the constituent publishes the city and state as a single output. Hotspot Finder is implemented as an occasion of the Eclipse SWT Browser.

III. Building a System

The **Amalgamated Application** expresses a perspective of software engineering that defines an application built by combining multiple existing functions into a new application. This is similar to mash up which is combining data and services to be applied to new kind of resources. However, amalgamated applications use business sources (e.g., existing modules or even Web services) of information, while mash ups usually rely on web-based. An amalgamated application consists of functionality drawn from several different sources. The components may be individual selected functions from within other applications, or entire systems whose outputs have been packaged as business functions, modules, or web services. All types of client and server applications were not initially designed to interoperate are gaining popularity. One of the reasons for this popularity is the competence to quickly reconfigure an amalgamated application for a task at hand, both by changing the set of mechanism and the way they are consistent. Service-Oriented Architecture (SOA) is a popular platform for building such amalgamated applications with the integrated mechanism being provided as Web services. A key curb of solely Web-service-based amalgamation is that it requires extra programming efforts when integrating non-Web service components, which is not cost-effective. Moreover, with the emergence of new standards, such as Open Service Gateway Initiative (OSGi), the mechanism used in amalgamated applications has grown to include more than just Web services. Our work enables progressive symphony of non-Web-service-based applications such as port lets, Web applications, native widgets, legacy systems, and Java Beans. Further, we planned a novel application of semantic annotation together with the standard semantic Web matching algorithm for finding sets of functionally equivalent applications out of a large set of available Non-Web-Service-based applications. Once such a set is identified, the user can drag and drop the most suitable constituent into an Eclipse-based symphony canvas. After a set of mechanism has been selected in such a way, they can be connected by data-flow arcs, thus forming an integrated, amalgamated application without any low-level programming and integration efforts.

Amalgamated Applications

The amalgamated Applications are being built in enterprises by assembling and representing business data and processes from multiple existing and related systems as unified and composite wholes. However, these solutions do not have to be restricted to composing Web services and information feeds; instead, they should include applications at higher levels of abstraction and user-centricity, such as structured presentation artifacts, unstructured artifacts as documents and spreadsheets with embedded business semantics, workflows, business rules, alerts, user-comprehensible business-intelligence, data models, prescanned reports, and data views. Symphony is a tool which has several languages, so that documentation creation and manipulation of data is minimized. The notion of situational applications is another interesting advent of relevance to the notion of solution symphony. A situational application is created on demand for a small number of users who have specific needs, and the duration of the application's existence is short-lived, similar to the trial version. Even though these types of applications are highly valuable to users in enterprises, they constitute a significant chunk of the "Long Tail" of user requests that are low on the list of enterprise IT priorities. Situational applications are a common use case for collaboration platforms, such as Microsoft Office Share Point—for example, an Office Share Point team-collaboration site that is created to support a new product campaign/launch. Situational applications provide immense value for the duration of the project.

IV. Package Pulverization

Packages are containers for classes. It is just like header files in C, C++ and is stored hierarchically. Package contains the classes for primitive data types, strings, math functions, threads etc. To pulverize the package, the cohesion and coupling ratio has to be calculated.

$$\text{Coupling (C)} = 1 - \frac{1}{d_i+2*c_i+ d_0+2*c_0+g_d+2*g_c+w+r}$$

d_i - Number of i/p data parameters
c_i - Number of i/p control parameters
d_0 - Number of o/p data parameters
c_0 - Number of o/p control parameters
g_d - Number of global variabl used as data
g_c - Number of global variabl used as control
w - Number of modules called (fan-out)
r - Number of modules calling the module under consideration (fan-in)

Coupling C makes the value larger the more coupled the module is. This number ranges from approximately from 0.67 (Low coupling) to 1.0 (Highly coupled).

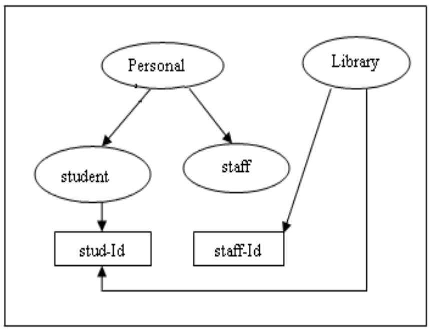

Fig 2. High Cohesion Representation

Fig 2 represents Personal class is inherited by student & staff class. Library Class uses staff-id & stud-id which are members of staff & student class. Number of related components = 2. So we can split the components. Fig 3 represents the splited components.

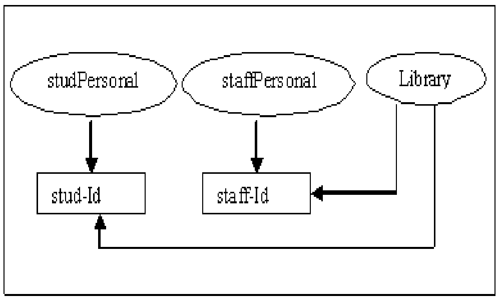

Fig 3. Low Cohesion Representation

If the package contains individual components, coupling ratio will be very low, so it is easy to pulverize it. It comes under the category of loose coupling. If the data is not complex means, we can separate the module easily. This will not have any impact on the module. If it is tightly coupled, the complexity of decoupling is very high.

Module Pulverization

It is a separate unit of software. Typical characteristics include portability interoperability, and language independent. Modules contain sub modules, member functions, data members. Inter relationship among all those things to be calculated using coupling ratio. Within module examine whether it is using global function or global variable. If it uses those it comes under context coupling. So to pulverize it, reference of the global variable or a function to be revoked without causing any impact for referencing sub module, functions and variables.

V. Data Member Pulverization

Data member is a variable or function used within a module or sub modules or packages. The member variable should not be similar in module or sub modules or packages and the scope of the variable and member should not be conflicted. If it is similar any one of the variable is renamed based up on their utilization inside the function or sub module. Ensure variable independency. If the variable is dependent reference has to be revoked without having an impact on referencing variable or function. It comes under the category of data coupling, so it can be decoupled easily. Decoupling is to reduce dependencies between the subsystems and its clients you want to decouple the subsystem and its data members as good as possible.

Algorithm:

```
        Package Lib;
/* represents the book details. It takes two situations when lib is empty and it contains the book.*/
Public class library implements Ilib
{
   /* Ilib is an interface
  If library is empty */
    Public int getBookDetail()
   {
       return 0;
     }
}
Package Lib;
/* represting non empty list */
Public class Nlibrary implements Ilib
{
    private object name,isbn,author,price;
    Private Ilib end;
  public object getBookDetails()
{
     Object obj[ ];
          obj[ ]  = end.name;
          obj[ ] = end.isbn;
          obj[ ] = end.author;
          obj[ ] = end.price;
       return obj;
}
}
```

Amalgamated Application

Package is pulverized into independent module, sub modules, and data members. The pulverized component will be amalgamated by creating ontology for individual components. To create an ontology protege tool has to be used. Inside the protege tool, the package classes, application classes, function classes, data member classes are all comes under root class "Thing". The package class contains name of the package that have been pulverized. User can select any of the packages he wishes to use in his application by selecting the name of the package, which is a subclass of package class. Once user selects the package the related functions, member variable, individuals will appear in forthcoming classes.

We selected the applications of School, College, and the application of library maintenance. In this, package contains school, college, and library. Modules contain personal details, book details. Personal details contain both student and staff details. Library detail contains details about books, journals, magazines and number of volumes. Functional manipulation depends on member variable. Addition function performs addition of books, journals, magazines in case of library package. In case of school package and college package addition function performs adding new staff details, new courses and student details. Likewise all operation has to be performed.

Sample figure 4 is presented for reference.

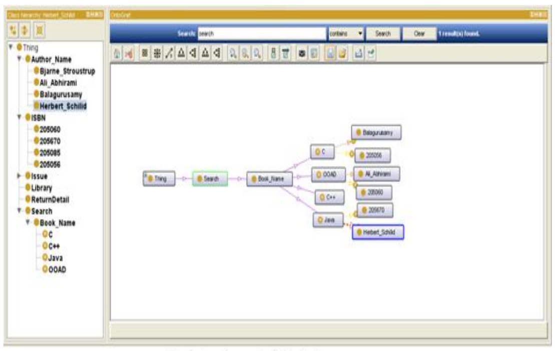

Fig.4.Amalgamated Ontology

VI. Conclusion and Future Work

The Users are facing difficulty in using large catalog. By using Eclipse based Symphony Canvas tool we can rectify the complexity like different environments, technologies, languages, times. By grouping all applications together we can reduce the complexity for the user. By using coupling and decoupling we can amalgamate and pulverize the applications whenever the user request. If the objects are loosely coupled, it is easy to pulverize the application, because changes made on applications has no impact. In tight coupling there is a problem were complex component cannot be easily separated, because of ripple effect.

Ripple effect is a situation which is like the ever expanding ripple across water, when an object is dropped into it, an effect from an initial state can be followed outwards incrementally. Ripple effect can be rectified by blum complexity axiom, because it satisfies running time and memory usage.

References

[1] H. Peter, H. Sack, C. Beckstein "SMARTINDEXER – Amalgamating Ontologies and Lexical Resources for Document Indexing".
[2] Martin Hitz, Behzad Montazeri "Measuring Coupling and Cohesion in Object-Oriented Systems".
[3] Timothy A. Budd "An Introduction to object-oriented Programming".
[4] Shymam R.Chidamber, Chris F.Kemerer "Towards a metric Suite for Object-oriented Design".
[5] Johann Eder, Gerti Kappel, Michael Sc hrefl "Coupling and Cohesion in Object-Oriented Systems".