Process of Assembling Component Based on Domain Ontology

Yan Wang
College of Mathematics and Computer Science of Bijie University, Bijie, China
Email: 604788055@qq.com

Xiufeng Cao
Department of Electronic and Information Engineering of Bijie Vocational and Technical College, Bijie, China
Email: 303363858@qq.com

Abstract—This paper introduced the related knowledge of domain ontology, analyzed the present situation of assembling component and the steps, and gave out a new assembly process of Domain Ontology Based Assembling Process (DOBAP). In addition, it introduced the overall framework of the system. In addition, System component classification engine based on domain ontology, component matching retrieval and component interface matching verification module are explained in detail. What’s more, traditional component retrieval and matching algorithm of domain ontology based on domain ontology is also improved.

Index Terms—Component Assembly, Domain Ontology, Component Retrieval, Component Matching

I. INTRODUCTION

Component assembly technology is one of the core technologies of component based software development. In software development, components development and assembling which are implemented by object-oriented technology is a popular method currently. These components are made up of objects, and in order to make the objects communicate with each other mutually by triggering, maintenance of static class and interface information is very necessary. This will lead to implicit dependencies, and also makes the objects depend on external services, such as communication middleware. These dependencies make the components difficult to be transferred between different platforms, and make the components assembly becomes more difficult. However, problems never exist in a unified platform.

The method based on the bus can avoid the assembly mismatch problem as possibly by strictly limiting the form of component in the system. Generally, definitions of component interfaces, exchange of data entity, interactive mechanism between components and the architecture style are explained clearly. Method based on bus is suitable for system construction of specific applications and architecture. Its ability of solving assembly mismatch problem is limited, and the scope of its application is small.

In the component-based software development, in order to solve the data exchanges between components, SUN released InfoBus for dynamic data sharing, which accords with 100% pure Java certification standards. It allows dynamic data exchange between the components by defining few interfaces and the statement of these interface protocol. Applet in the HTML page or JavaBeans in JBuilder environment can interact with InfoBus. InfoBus also can be used in any Java class, such as Applet, Servlet, and so on [1-3].

Component-based systems are made up of collection of interacting entities called components. The idea in component-based software engineering (CBSE) is to develop software applications not from scratch but by assembling various library of components. This development approach allows to extend component-based systems via plug and play components and to reuse components. Therefore, one saves on the development costs and time. A component is a unit of composition with contractually specified interfaces and explicit dependencies. An interface describes offered and required services by a component without disclosing the component implementation. It is the only access to the component information. Interfaces may describe component information at signature, behavior or protocol, semantic, and quality of services levels. The success of applying the component-based approach depends on the interoperability of the connected components. The interoperability can be defined as the ability of two or more entities to communicate and cooperate despite the differences in their implementation language, the execution enviornment, or the model abstraction. The interoperability holds between components when their interfaces are compatible [4-6].

However, the research of automation assembly is still in the exploratory stage. One of the main difficulties is how to describe the various features of components, so that the assembly system can understand and to select the component according to these characteristics, and then assembled into application system. Ontology is a concept model that can describe system in the semantic and knowledge level. Its purpose is to obtain knowledge in a general way, and to provide common consensus for domain concepts, thereby enhancing the processing ability of knowledge. It has been widely used in search field in recent years and a lot of achievements have been
reached. Therefore, the DOBAP introduces ontology into component assembly, and to describe the component by using the knowledge concept model of domain ontology. In addition, it retrieves and matches components by using retrieval algorithm based on ontology, constructing application system automatically.

II. ONTOLOGY

Ontology is a clear instruction for shared conceptual model. In the field of computer, its role is to get generic knowledge, and to provide consensus understanding. It defines domain concepts formally, and also stressed the linkages between these concepts.

Ontology literally means “the study of the state of being.” In a formal sense, it is a description of what things are in a particular model. Formal ontologies are too often associated simply with taxonomies of classes, but in more expressive formal systems they also include general structural descriptions, constraints, and axioms. A meta-model is a type of formal ontology. We choose to employ the term formal ontology here, however, because its association with knowledge representation and reasoning helps differentiate our efforts from the more traditional ADLs which, to our understanding, are not usually intended to support automated reasoning.

Our ontology identifies three basic types of architectural elements: component, interface, and service. Based on our observations, we chose to make connector a type of component, and differentiate functional-component from connector below the component concept. We further formalize the differences between a functional component and a connector below, however there is some ambiguity left, and it is up to the architects to determine when a component is a functional component and when it is a connector. The primary focus we have is on reusing knowledge about integrating functional components. The notions of services, interfaces, and connectors are therefore central to our ontology, as these concepts represent the specifications for how functional components integrate. In addition to the architectural elements, we also define group and project as types of objects that we will need to associate with particular instances of the architectures we create. An instance of project will represent a single instantiation of an architecture. In other words, each time an architecture is used, there will be a single project object associated with it. The project object is needed by our document generation system to provide a single point of reference into any architectural model. We take the position that formal ontology should not be done without specific goals in mind. Our ontology goals are to develop a basic ontology for representing architectures, equivalent to ADLs with the addition of automated reasoning. Architects then specify architectures using the elements of our formal ontology. Completed architectural models are then instantiated for individual projects. This instantiation is characterized primarily by the creation of several documents that describe the architecture and specify how it will be used to solve the particular problem the project is concerned with [7-10].

A. Ontology Construction

Domain ontology model mainly reflects the relationships between concepts in the domain. Ontology provides explicit definitions of terms, making the relationship between concepts very easy to be understood and to be solved. Ontology used in this paper is constructed by Protégé based on artificial intelligence of Shanghai component library. First of all, it lists all the concepts in the field, and then makes definitions of these concepts. The basic description information of the components in artificial intelligence field mainly consists of function types, development language, operating environment, operating system, development status, oriented users, license and internationalization, and they all have their own term spaces. For example, text processing, scientific research and education, program development, gaming and entertainment belongs to component function type, while non functional type mainly includes the development of language, operating environment and operating system [11]. Secondly, it describes relationships between concepts, such as the development of languages including Java, C and C++, which correspond to kind-of relationship. Construction of ontology is shown in Figure 1.

![Figure 1. Ontology Architecture](image)

OWL description of one class is shown as follows:

```xml
<owl Class rdf:id="Windows NT">
<rdfs: subClassOf rdf:resource="#running environment"/>
</owl: Class>
```

The software architecture community has been touting the advantages of a higher level approach to reuse for the past few years (Shaw and Garlan, 1996). Where synthesis systems require a complete, formal (and correct) axiomatization of a domain in order to realize reuse, architectures can be reused fairly easily. The architecture exists at such an abstract level that most details have been removed. The connection to software is sometimes unclear, and precisely what gets reused is also quite vague: it may be pictures, documents, parts of documents, software components, parts of software components, off-the-shelf software packages, etc.

In all cases, from synthesis to architecture, it is clear that knowledge does get reused. The difference is the amount to which that reuse can be reflected in resource savings, and the degree to which the knowledge can be transferred to others.

We have been focusing on reusing knowledge about software architectures through documents. Our goal is to
capture knowledge about a specific “solution” architecture in such a way that each time an architecture is used to solve a problem, the documents associated with it (such as statements of work, design descriptions, etc.) can be generated automatically with minimal effort. Given a sufficiently expressive tool for representing the knowledge associated with architecture, we have found a formal ontology to be a critical enabling technology for this work [12-14].

The domain ontology is used for a specific ontology in designated fields. It gives a formal characteristics and laws of the domain entities concept and the relationship between the activities in this field with description.

A domain ontology is defined as \( O \lessdot \{C, R, M, A, I\} \). \( N \) represents ontology, \( C \) represents the concept in the field, \( R \) is the relationships between concepts, \( M \) is method, \( A \) represents the concept of attribute set, and \( I \) is the set of all axiom. The relationships between the concepts include the following:

1. Association
   Association relationship is used to describe the equivalence relationship between concepts. Association (Cl, C2) represents the concept \( C_1 \) associates with \( C_2 \).

2. Is - A
   Is - A relationship is a typical relationship used to indicate things with abstract generic relationships. It formed a logical hierarchy between concepts. is - A (\( C_1, C_2 \)) represents \( C_1 \) belongs to the class of \( C_2 \).

3. instance - of
   The relationship instance – of is a typical relationship between concepts and individuals. For the concept \( C \) and its example set \( S_e \), relationship between element e in instance set \( e \in S_e \) and the concept \( C \) is called instance relation, denoted as instance \((e, C)\), which is explained as \( e \) is an instance of \( C \) in semantic interpretation.

4. has - A
   has - A relationship is between the whole and part of concepts. has - A \((C_1, C_2)\) denotes \( C_1 \) is a part of \( C_2 \).

B. Matching Regulation of Ontology

Match the collection of a word to the domain ontology, a generally accepted concept set is obtained. The document collection of a component can be expressed as \( T = \{t_1, t_2, ..., t_n\} \). Match the set \( T \) to domain ontology, and a concept set \( C = \{c_1, c_2, ..., c_m\} \) can be got. Domain ontology can be described by OWL, and the matching rules are as follows:

1. When \( t_i \ (i = 1, 2, ..., n) \) in set \( T \) could be matched with the concept in the ontology, put the concept \( c_j \ (j = 1, 2, ..., m) \) to concept \( C \).

2. When \( t_i \ (i = 1, 2, ..., n) \) in set \( T \) not only can match with the properties of concept in ontology, but also can match with other concepts or the instances of concept, the subsequence of priority is concept, instance and properties. Then output the corresponding concept as matching concept, otherwise, add the corresponding concept to the set \( C \).

3. When \( t_i \ (i = 1, 2, ..., n) \) in set \( T \) only match with an instance of ontology, the corresponding concept is added to the set \( C \).

4. When the element of set \( T \) does not match with any concept, discard it. Now the word has been obtained by matching with the domain ontology. This paper defines similarity formulas between two concepts. Properties of concepts \( C_1 \) and \( C_2 \) are \( C_1 = \{t_1, t_2, ..., t_m\} \); \( C_2 = \{t_1', t_2', ..., t_m'\} \).

If \( \{t_1, t_2, ..., t_m\} \cap \{t_1', t_2', ..., t_m'\} = \{p_1, p_2, ..., p_k\} \) and the weight of each concept in set \( p \) is defined as \( \{w_1 + w_2 + ... + w_k\} \), similarity between the two concepts is

\[
Sim(c_1, c_2) = (w_1 + w_2 + ... + w_k) / k
\]

Similarity between two concepts can be got by computing the average value of the sum of common attributes weights [15-17].

C. Description of VSM Model Based on Ontology

In order to get a feature vector of each text, text set is processed in advance, and the obtained feature vector is mapped to domain ontology to get a new feature vector. First, text set is divided through IKAnalyzer. Each text can be denoted as \( d \), \( d = \{t_1, t_2, ..., t_m\} \) \((i = 1, 2, ..., m)\), and \( t_j \ (j = 1, 2, ..., n) \) is matched with domain ontology to get a new vector \( Ont \_di \ = \ (c_1, c_2, ..., c_n) \). In traditional vector space model, text can be expressed as the following vector:

\[
IV(d) = \{w_1d_1; w_2d_2; ... ; w_nd_n\}
\]

In the formula, \( t \) signifies word items, and \( w(d) \) is the weight of \( t \) in \( d \). \( tfidf \) is a common computing method of word weight used in this paper. In addition, a text can be expressed as a vector of \( n \) dimensional space. \( V(d) = (w_1, w_2, ..., w_n) \) is called as space vector model. Computing of weight in each word item is as follows:

\[
\text{fidf}(d,t) = tf(d,t) \times \log(|D| / df(t))
\]

D denotes text set, \( d \) signifies arbitrary text, \( t \) is the word in text. \( df(t) \) is discovery frequency of word \( t \) in text \( d \), \( D \) is the number of text set, \( df(t) \) is the frequency of occurrence of word \( t \) in text set, and \( tfidf(d,t) \) is the weight of \( t \) in text \( d \).

III. PROCESS OF COMPONENT ASSEMBLY

Component assembly in application system based on component can be divided into 3 stages.

1. Automatic construction of framework model in application system by the formal description for requirement is carried out in requirement engineering,
e.g., assembly model based on TLG. Because formal description of demand is involved in this stage, which is a difficult problem in software engineering, study in this phase got no great breakthrough in recent years.

Figure 2. Component assembly process

(2) Component retrieval, instantiation of application framework model, and generation of component framework of application system. This stage can be divided into two substages. In the first stage, matching component is automatically retrieved according to component description in framework model of application system, and component connection describing is generated, e.g., JohnPenix’s Rebound model. The core idea is to choose self-adaptive strategy of component through matching component specification with requirements specification. However, because of formal description of specification and its poor actual operation, it is difficult to be used in practical application.

Solving mismatching problems between components by using various automated processing technologies in system framework is the second stage, such as connection model proposed by Bridget Spit-znagel and Zhong Wang’s MAC technology.

(3) Extract components from component library, generate deployment description documents which are adaptive to specific application platform, and automatically deploy application system. For example, component deployment files on J2EE platform output by ABC assembly tool. Although automated assembly of components can be divided into 3 stages, there are no constraints between every two stages, i.e., missing automation in any stage has no effect on the automatic realization of the next stage. The part which is not implemented automatically can be manually solved by system development personnel [18-20]. See Figure 2.

IV. PROCESS OF DOBAP

A. DOBAP Overview

DOBAP is based on the CBD method guided by domain architecture. Domain architecture is a software architecture which is closely related to application domain. It provides basic structure and operations for domain problems, and provides a method of extending or re-development to meet the special demand for specific application. Mature domain architecture includes MetaH embedded system structure belonging to IBM San Francisco and Honeywell.

DOBAP is mainly composed by graphic modeling interface of application system framework model, domain ontology and its administrative tools, component library and its management system integrated with component classification engine, component retrieval and matching module, components interface matching authentication module and ADL script generator. Structure diagram is shown in Figure 3.

B. Fabrication of Component

Component-based development usually includes component acquirement, component classification and retrieval, component evaluation, adaptation and constructing a new system with existing components in the new context. There are many different ways to get components.

(1) Obtain required components from existing components, and directly use it or do some adaptation to get reusing component;

(2) Extract components which have potential value of reuse through the legacy project to get reusable component;

(3) Purchase commercial components on the market, namely the COTS component;

(4) Develop new components to meet the requirements. During the decision of above strategies, one-time costs and maintenance costs must be considered.

Regardless of the ways of acquisition, the process of making components should be considered. We discuss the production of reusable component and assembly process issues with C++ language.
C. Graphical Modeling

Graphical modeling is visual editing environment of application system framework provided by DOBAP for developers. The first job of component assembly is to construct framework model of application architecture. The whole framework model consists of several interrelated component searching template, and the connections reflect call relation between the interfaces. Developers create template of component retrieval in this editing environment according to domain architecture, establish connections, and specify the retrieval and matching properties, so as to establish framework model of the whole architecture. The information of component template mainly includes component domain features, as well as interface functions and connections.

D. Component Retrieval and Matching Based on Ontology

Cluster parameters needs to be pre-defined in traditional K-Means algorithm, i.e., the value of K should be confirmed. Different value has different clustering results. However, the algorithm in this paper does not rely on the value of K. Set the first component as cluster point, and compute the similarity of each component. A threshold value simArg is need to be set. When sim(d_i, c_j)≥simArg, put it in the same cluster. On the contrary, set this component as the second cluster point, and compare the other components with it, and the like. The algorithm is described as follows:

1. Initialization. Component text information were cut respectively and stored in component array by using word segmentation technology;
2. Loading Rules. All rules in rule base are loaded into memory, using XML to store. Rules are expressed by four tuples, i.e., R(rf, r, rsf, rs) in which R signifies rules, rf denotes reasoning assumes facet, r expresses reasoning hypothesis, rsf signifies reasoning conclusion facet, and rs denotes reasoning conclusion;
3. Ontology Semantic Expansion. Extend the semantic of ontology according to the loading rules, making a term extend into a collection of terms;
4. Ontology Reasoning. Reasoning is based on three axioms SubClassOf, EquivalentClasses, DisjointClasses of Protégé;
5. Set the first component as cluster point c_j, sim(d_i, c_j) between c_j and other components were calculated. In addition, a threshold simArg is set, and when the sim(d_i, c_j)≥simArg, put them together in a cluster. Conversely, make the current member as the second cluster point, respectively calculate the similarity of other components, and so on;
6. Clustering. Finally, put some small clusters together, separate the class and carry out special treatment.

Therefore, the greatest advantage of this algorithm is that the cluster number need not be predefined. However, some small classes may be produced after clustering. For example, some classes may only contain one or several components. Therefore, the small classes also need to be optimized and combined, in order to get better clustering results [21].

Component clustering algorithm based on ontology is compared with K-Means algorithm based on vector space model, and the results proved that the former algorithm is effective. Clustering result can be seen in this experiment that two different concept description of two components are clustered in the same group, which fully reflects the role of ontology in semantics. Experiment data come from AI components of Shanghai component base, and three parameters clustering precision, clustering recall, and F-Score are given as follows:

Suppose there are k classifications in clustering j, then number of component in classification i is

\[ N_q = N_j - N_{ij} - N_{kj} - \ldots - N_{(i-1)j} - N_{(i+1)j} - \ldots - N_{ij} \]  (4)

Clustering Precision (P):

\[ P = \frac{\text{Precision}(i, j)}{N_j} = \frac{N_q}{N_j} \]  (5)

Clustering Recall (R):

\[ R = \frac{\text{Recall}(i, j)}{N_j} = \frac{N_q}{N_i} \]  (6)

F-Score Method:

\[ F - \text{Score} = \frac{2 \times P \times R}{P + R} \]  (7)

The three parameters are respectively 55%, 60%, and 58% of K-Means algorithm which is based on vector space model, while that of clustering algorithm based on ontology are respectively 79%, 70% and 75%. The experimental results suggested that domain ontology can improve the quality of clustering and further enhance the efficiency and accuracy of component retrieval.

E. Interface Matching Verification of Components

After the above process of component retrieval and matching, component retrieval template in framework model of application system has been basically instantiated, and has been replaced by actual component description model. However, above matching is just of function semantic, while matching of interface connections on implementation layer is also need to be verified. In this phrase, matching idal of JeannetteM. Wing in Canaki Mellon University is adopted. JeannetteM. Wing gave the following definition of strict matching on implementation layer.

\[ \text{Match}(C, C') = \text{mspec}(\text{spec}_m, \text{spec}_c) \wedge \text{msig}(\text{sig}_m, \text{sig}_c) \]  (8)

If the signature of two components C and C' match to msig, and the specification match to mspec, the two components are matched. According to the definition, symbol matching verification on implementation layer requires both signature matching and verification. Signature matching is relatively simple, the names of the two interfaces are compared with each other in this process. However, specification matching is more important. Although Jeannette M. Wing has provided a set of describing and matching method, the description is based on a formal method of first-order logic, which is
difficult to describe in practical application. Therefore, the semi-structured language XML is adopted to describe interface protocol of components [22-23].

V. CONCLUSION

This paper analyzed the each stage and aspect of component assembly, and gave a process DOBAP based on domain ontology supporting automatic assembly of components. Traditional component retrieval and matching algorithm was improved, and structural description methods were adopted in the aspect of component interface functions and interface specification, resulting in simplifying the complexity of component interface matching. In the system using, in order to connect to the practical application, the system sets aside interface matching. In the system using, in order to connect to the practical application, the system sets aside interface matching. In the system using, in order to connect to the practical application, the system sets aside interface matching. In the system using, in order to connect to the practical application, the system sets aside interface matching. In the system using, in order to connect to the practical application, the system sets aside interface matching.

ACKNOWLEDGMENT

The work is finished under the help of my friends and colleagues. Thanks for their help.

REFERENCES


