Proceedings of the 17th World Congress
The International Federation of Automatic Control
Seoul, Korea, July 6-11, 2008

Components Selection Methods for Enterprise Interoperability in Multi Domain Models

Ke FENG, Xiaoping LI, Qian WANG, Jingjing SHAN

School of Computer Science &Engineering, Southeast University, 210096, Nanjing, P.R .China
Key Laboratory of Computer Network and Information Integration (Southeast University), Ministry of Education
alphafeng2006@gmail.com xpli@seu.edu.cn qwang@seu.edu.cn sjj93@126.com

Abstract: Component-based development is gradually showing its advantages in building complex systems with shorter time and less cost than traditional methods. However, mismatching and semantic are key problems in component searching process. In this paper, a function description model is proposed to precisely and completely describe the user requirements and component function based on domain models. In the component selection process, a sophisticated matching algorithm is introduced for semantic problems in matching two activity profiles derived from different domain models. A component selection method is also presented to improve interoperability for multi domain models, followed by the implemented prototype of the proposed methods.

1. INTRODUCTION

Component-based development (CBD) is increasingly changing the way of building new systems and showing its advantages. (1) Reduce the developing time and cost. (2) Design and deploy complex software systems with minimum engineering effort and resource cost. Although CBD has many potential benefits, some issues should be considered. (1) How to describe requirements for and capabilities of a component precisely and completely. (2) How to search the desired components in existing component repositories efficiently. The key issue is how to compare the user requirements with the capabilities of the components, which is a component selection process.

In the existing work, there are five main component selection approaches (Vijayan *et al.*, 2003).

Keyword Search: Search for the occurrence of string patterns specified by the user in component attributes and descriptions.

Faceted classification: Classify components based on facets (taxonomies) such as function the software performs, medium used, type of system, functional area, etc.

Signature Matching: Matching of function types and argument types to the query specified by the user. Signature matching could be one at the function level or module level (set of functions).

Behavioral Matching: Execute each library component with random input vectors and generate output vectors. Compare expected output to actual output and select components.

Semantic-Based Method: User requirements expressed as simple imperative or nominal sentences. Natural Language Processing (NLP) used for generating initial queries and augmented with domain information. Components selected based on closeness measure.

A component selection process is characterized by uncertainty, dynamic changes of the environment, explicit and implicit criteria and constraints and involving in different stakeholders (Günther, 2003). Different users and software vendors use heterogeneous methods to describe their requirements and component capabilities. These different descriptions focus on different aspects. Furthermore, different descriptions may also use different vocabularies for the same item. As a result, there are some semantic problems during the selection process and semantic-based method is usually used. Many researches focused on the semantic heterogeneity existing in many autonomously developed systems (Vijayan et al., 2003. Gemma et al., 2004. Sofien et al., 2006. Ayala et al., 2004). However, without considering the application context, semantic-based matching approaches appear not adequate for addressing the challenge (Hung-Ju et al., 2007).

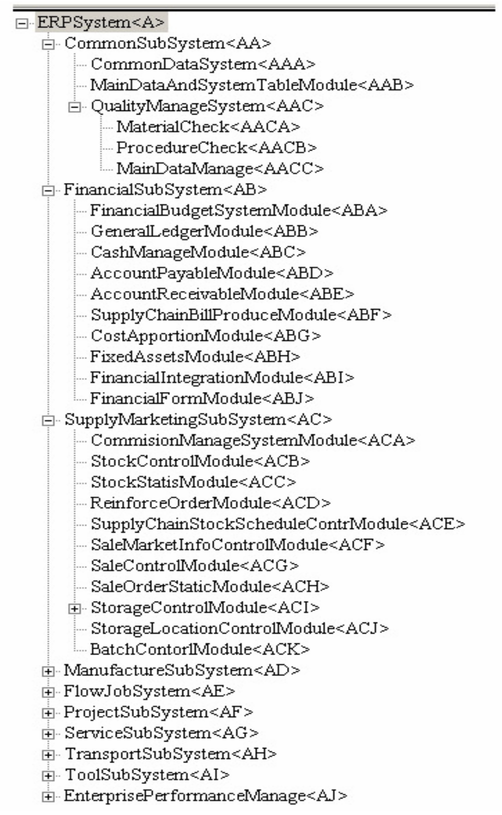
Besides the semantic problem, there are some problems on efficiency in the selection process. When the number of components is large, there are a lot of descriptions in the repository. So, it is essential to develop an efficient component selection procedure and the components should be well structured to reduce the search scope. Several approaches have been introduced to reclassify the software components and simplify the request specification (Sofien *et al.*, 2006).

Interoperability between different users and software vendors should be improved by a uniform and standard way to describe user requirements and component capabilities precisely and completely (Xavier *et al.*, 2005). These different descriptions should also use common unified items to avoid the misunderstanding about the described capabilities between each other (Nasib, 2003). In this paper, basic concepts used in the domain model are defined for interoperability. With the domain model, the application context will be fully considered. Formal component and user requirements descriptions are extended with functional and non-functional aspects. As well, an algorithm is presented for semantic matching in component selection process. At last, a component selection method is introduced.

2. REFERENCE FRAMEWORK

2.1 Domain Model

In a domain model, an integrated application can be modeled as a combination of a set of processes, resources and information exchanged. Resources consist of networks, devices, software, equipment, material, and personnel necessary to support the processes and information exchange required by the application (ISO 16100-1, 2002. ISO 16100-2, 2002. ISO 16100-3, 2005).



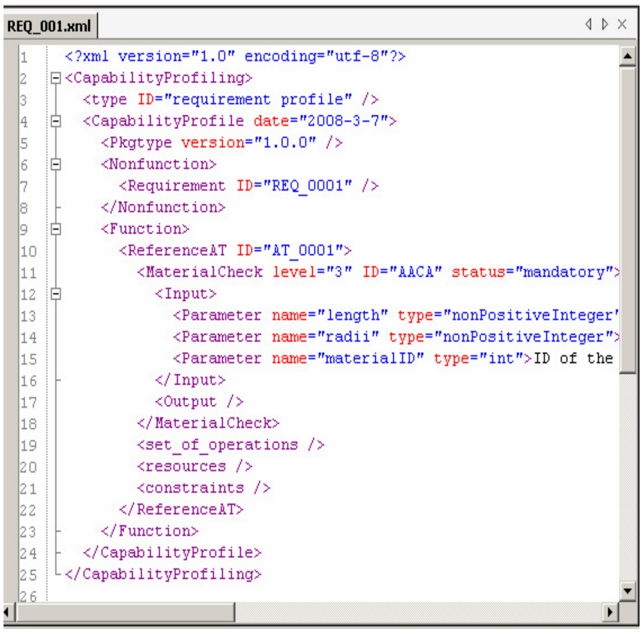
```
ERPSystem<A>
  CommonSubSystem<AA>
    CommonDataSystem<AAA>
    MainDataAndSystemTableModule<AAB>
    QualityManageSystem<AAC>
      MaterialCheck<AACA>
      ProcedureCheck<AACB>
      MainDataManage<AACC>
  FinancialSubSystem<AB>
    FinancialBudgetSystemModule<ABA>
    GeneralLedgerModule<ABB>
    CashManageModule<ABC>
    AccountPayableModule<ABD>
    AccountReceivableModule<ABE>
    SupplyChainBillProduceModule<ABF>
    CostApportionModule<ABG>
    FixedAssetsModule<ABH>
    FinancialIntegrationModule<ABI>
    FinancialFormModule<ABJ>
  SupplyMarketingSubSystem<AC>
    CommisionManageSystemModule<ACA>
    StockControlModule<ACB>
    StockStatisModule<ACC>
    ReinforceOrderModule<ACD>
    SupplyChainStockScheduleContrModule<ACE>
    SaleMarketInfoControlModule<ACF>
    SaleControlModule<ACG>
    SaleOrderStaticModule<ACH>
    StorageControlModule<ACI>
    StorageLocationControlModule<ACJ>
    BatchContorlModule<ACK>
  ManufactureSubSystem<AD>
  FlowJobSystem<AE>
  ProjectSubSystem<AF>
  ServiceSubSystem<AG>
  TransportSubSystem<AH>
  ToolSubSystem<AI>
  EnterprisePerformanceManage<AJ>
```

Fig. 1.Activity tree structure

A process is modeled as a set of activities that follow a specific sequence. Each activity has its own specific function(s) and is performed by the corresponding component(s) offered by different vendors. As shown in Figure 1, all activities are organized as a tree. A parent activity can be decomposed into several child activities and every leaf activity is an atom activity. The activity tree (AT) shall be pre-defined by domain experts, and different domains may have different activity trees. An activity tree is a kind of domain model which actually indicates how the domain experts define the functions of each activity needed in the target domain and shows the part-whole relation of each activity in the function matching process. At a minimum, the activity tree defines the scope and boundaries of the essential standard components of a software system and can be used principally as an interface specification.

2.2 Activity profile

An activity profile (shown in Figure 2) is used to describe the component capabilities and user requirements. There are two parts in a profile. (1) Nonfunction Part including general information of a component, such as type of manufacturing domain, reference activity tree ID, computing facilities, component performance and so on (W. Yu et al., 2007). (2) Function Part for the required/needed activities and particular information required by each activity, such as input/output information (to be exchanged), resources needed, lower level activities (functions), and so on. The format of Function Part is shown in Figure 2. An activity profile is a well formed XML file. In the component selection process, the function part of profile is our main concernment to evaluate the interoperability.



REQ_001.xml

```
<?xml version="1.0" encoding="utf-8"?>
<CapabilityProfiling>
  <type ID="requirement profile" />
  <CapabilityProfile date="2008-3-7">
    <Pkgtype version="1.0.0" />
    <Nonfunction>
      <Requirement ID="REQ_0001" />
    </Nonfunction>
    <Function>
      <ReferenceAT ID="AT_0001">
        <MaterialCheck level="3" ID="AACA" status="mandatory">
          <Input>
            <Parameter name="length" type="nonPositiveInteger'
            <Parameter name="radii" type="nonPositiveInteger">
            <Parameter name="materialID" type="int">ID of the
          </Input>
          <Output />
        </MaterialCheck>
        <set_of_operations />
        <resources />
        <constraints />
      </ReferenceAT>
    </Function>
  </CapabilityProfile>
</CapabilityProfiling>
```

Fig. 2.An example of activity profile

3. FCUNCTION MATCHING

Activity should be matched to find existing proper components which meet the user requirements in the component selection process. In the matching process, the profile of user requirements is compared with the profiles of existing components in Database one by one. So the activity matching is actually a matching process between activity profiles, which can be derived from any two nodes of activity tree(s).

The matching includes two parts due to the above activity profile structure. Non-Function Part has a fixed information structure and contains the general information of an activity. So the matching process can be fulfilled just by comparing all the elements one by one.

The contents of Function Part (shown in Figure 2) are mainly derived from activities of target domain. When component and user requirements base on the same domain model, the matching process is Simple Function Matching. When component and user requirements have different domain models, the matching process is Sophistical Function Matching, which is much more complex than the simple one. In this paper, we focus on Function Matching

3.1 Simple Function Matching

In an activity tree, a parent activity always consists of its child activities. Therefore, in the CCS shown in Figure 3, the component (node a1) includes the function of the user requirements (node a3). In other words, the user requirements can be fulfilled by the component, and a proper component has been found.

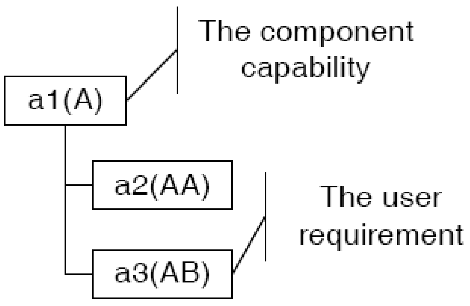


Fig. 3. A simple function matching example

In simple function matching process, LID (Level_ID) is used to indicate the location of each node in the same activity tree. LID of an ancestor node is always the prefix of those of its descendants. For instance, A and AB are LIDs of a1 and a3 respectively. A is the prefix of AB, which means a1(A) is the ancestor node of a3(AB). LID implies the level of each node and the function inclusive relationship of two selected nodes in the same activity tree.

3.2 Sophistical Function Matching

Different organizations may have different activity hierarchical structures and naming rules in certain domain, leading to various activity trees. Therefore, semantic problems are the main consideration when component and user requirements come from different activity trees (Andrea *et al.*, 2003). In this paper, 'semantic' means the meaning of an activity name and the function represented by the activity. Different activity names stand for the same function (show as Figure 4(3)), or the same name stands for different capabilities (show as Figure 4(2)). Also, activity may have different names with different capabilities (show as Figure 4(4)).

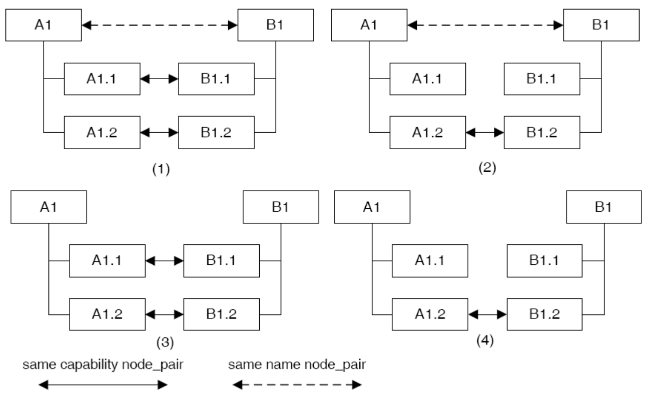


Fig. 4. Semantic problems between different ATs

Therefore, LID and name comparison used in the simple function matching is unsuitable for CCS structure comparison, which is necessary for the semantic problems in Sophistical Function Matching.

3.2.1 Same Function Node_pair

CCS_A, CCS_B are two CCSs, $a \in AT\_A$, $b \in AT\_B$, $AT\_A \neq AT\_B$.

Definition 1 (same function node_pair): a and b are called the same function node_pair (marked as $a == b$) when the following two conditions are satisfied. (1) At least a or b is a leaf node. (2) a and b have the same name according to the synonymy dictionary.

In an activity tree, leaf nodes are atomic activities and perform limited functions. Hence, it is not difficult to find the same function node_pairs in leaf nodes even though different organizations may use different activity trees for the same domain, especially when the domain synonymy dictionary is used. The same function node_pairs provide basic rules for the structure comparison of different activity trees.

3.2.2 Function Satiable

Definition 2 (function satiable): b is function satiable to a (marked as $b \to a$), if and only if: $a == b$ or descendants of b are function satiable to all child nodes of a.

Definition 3 (corresponding node): if b is function satiable to a, then b is the corresponding node of a.

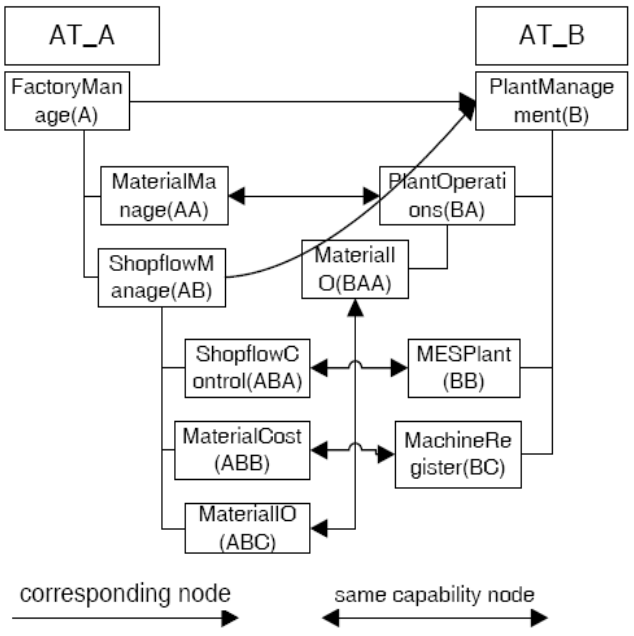


Fig. 5. An example of function satiable

For an instance shown in Figure 5, Node A is user requirements in the left-hand tree. A suitable node should be found to make node A function satiable in the right-hand tree.

(1) Find the corresponding node of A by Definition 3.

(2) Recursively find the corresponding nodes of the child nodes of A (AA and AB).

(3) $AA == BA$.

(4) The child nodes of AB (ABA, ABB and ABC), $ABA == BB$, $ABB == BC$ and $ABC == BAA$. Ancestor node of BB, BC and BAA is B, so $B \to AB$.

(5) $B \to A$ as the ancestor node of BA ($AA == BA$) and B ($B \to AB$) is B.

3.2.3 Algorithm FCN

In this paper, algorithm FCN (Find Corresponding Node) is presented for finding corresponding nodes.

(1) Scanning AT_A to find all same function node_pairs.

(2) 1.1 Set the default state =-1 for each node $a \in AT\_A$.

1.2 Scan each node $a \in AT\_A$

```
{
  if (a == b)
    set a.state=2 and store (a, b) in the mapping table;
  else if (node a is a leaf node)
    set a.state = 0;
}
```

(3) Scanning AT_A to find the corresponding nodes for all the non-leaf nodes in AT_A. For node $a \in AT\_A$, call FindCN (a) recursively by the following way.

```
{
  if( a.state > 0)
    return a's corresponding node;
  if(a.state == 0)
    return null;
  if (a.state ==-1)
  {
    //i in (1.. the node number of a's child nodes)
    for each node $a_i$ in a's child nodes
    {
      node $b_i$ = FindCN($a_i$);     // recursive call
      if ($b_i$ == null)
        set a.state = 0 and return null;
    }
  }
  get the common ancestor b of all $b_i$
  set a.state = 1;
  store (a, b) in the mapping table;
  return b;
}
```

(4) Stop.

For the example in Figure 5, the mapping table, which records the node state and corresponding nodes of the nodes in the left-hand tree, is shown Table 1.

Table 1. Node states and corresponding node_pairs

Node LID	step1		step2	
	node state	corresponding node LID	node state	correspondin g node LID
A	*-1*	-	*1*	*B*
AA	2	BA	2	BA
AB	*-1*	-	*1*	*B*
ABA	2	BB	2	BB
ABB	2	BC	2	BC
ABC	2	BAA	2	BAA

3.2.4 Algorithm Analysis

In step1, time is mainly consumed by the same function node_pairs searching process in terms of Definition 1. Suppose n nodes in AT_A and m nodes in AT_B. Comparing a node name in AT_A with the names in AT_B one by one, the time complexity is $O(m)$. If every node in AT_A has k synonymies at most in the synonymy dictionary, the worst time complexity of the same function node_pairs searching process is $O(k * m * n)$.

In step2, the recursive corresponding node searching process is a bottom-up approach. By using node state, all the nodes can be processed just once. In order to find the corresponding node of a non-leaf node a in AT_A, all corresponding nodes of its child nodes should be found recursively. Then, the corresponding node of a can be quickly found just by comparing the LIDs of the corresponding nodes of its child nodes. For example, LID BB and LID BC have the same prefix B. So, the common ancestor of BB and BC is B. The time complexity of finding the same prefix is $O(1)$, and the time complexity is $O(m)$ by using this LID (the same prefix) to search the node in AT_B. Therefore, the time complexity of searching the corresponding nodes of all the nodes in AT_A is $O(m * n)$.

In order to decrease the time expense of FCN, two hash tables are used to accelerate the searching process in AT_B. One is used to index and link the nodes in AT_B by their names. Suppose the hash table has p hash buckets, the average time complexity for searching an element in the hash table is $O(m / p)$. So, in step 1, the time complexity for searching a node by its name in AT_B can be reduced to $O(m / p)$. In step 2, the other hash table indexes the nodes in AT_B by their LIDs to search the corresponding node efficiently.

3.3 User Requirements Translation

With the mapping table, user requirements can be easily translated to the activity tree with registered components in certain domain (component AT for short). In Figure 5, suppose FactoryManage(A) is the user requirements, its corresponding node is PlantManage(B) based on the mapping table (show as Table 1), then the user requirements can be translated to PlantManage(B) in component activity tree. Based on the mapping table, the node in user requirements do not have corresponding node, use its ancestor's corresponding node instead.

4. MSU SELECTION

The MSU selection process can be modeled as Figure 6.

(1) User inputs the requirements.

(2) Get the domain, activity tree ID and selected activities based on the REQ.

(3) Search the components registered on the same function classes in Database (components and user requirements are on the same AT). If any mapping tables exist, translate the user requirements from original AT (AT with the user requirements) to new AT according to the mapping table. Then search the components on the new AT.

(4) Compare the components found in (3) with REQ.

(5) Output the components which meet the REQ.

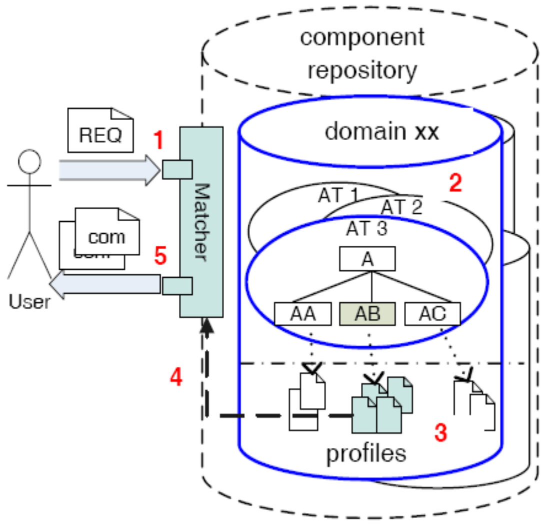
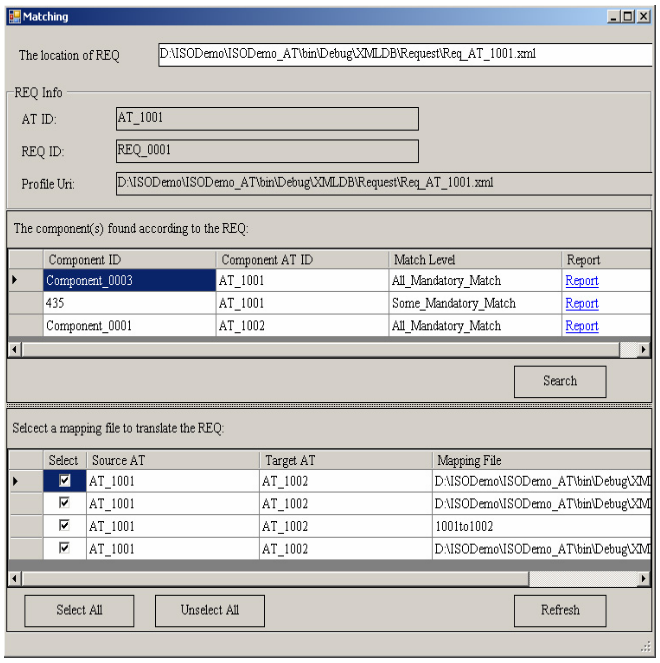


Fig. 6. Software selection process

The simple function matching process works well and efficiently in step (2). Algorithm FCN is used to generate the mapping tables used in step3.

A screenshot of our components selection application is shown in Figure 7. In the searching process, the application search and compare the components register on the original CCS. Then the application translates the user requirements from original AT to component AT according to the mapping tables saved in selected mapping files, and redo the searching and comparing process on the component AT.


Matching

The location of REQ: D:\ISODemo\ISODemo_AT\bin\Debug\XMLDB\Request\Req_AT_1001.xml

REQ Info

AT ID: AT_1001

REQ ID: REQ_0001

Profile Uri: D:\ISODemo\ISODemo_AT\bin\Debug\XMLDB\Request\Req_AT_1001.xml

The component(s) found according to the REQ:

Component ID	Component AT ID	Match Level	Report
Component_0003	AT_1001	All_Mandatory_Match	Report
435	AT_1001	Some_Mandatory_Match	Report
Component_0001	AT_1002	All_Mandatory_Match	Report

Selcect a mapping file to translate the REQ:

Select	Source AT	Target AT	Mapping File
✓	AT_1001	AT_1002	D:\ISODemo\ISODemo_AT\bin\Debug\XM
✓	AT_1001	AT_1002	D:\ISODemo\ISODemo_AT\bin\Debug\XM
✓	AT_1001	AT_1002	1001to1002
✓	AT_1001	AT_1002	D:\ISODemo\ISODemo_AT\bin\Debug\XM

Fig. 7. A screenshot of components selection system

5. CONCLUSIONS

In this paper, a components selection method based on multi domain models is proposed. An activity tree is a model of the certain target domain which classifies the functions and guides the components selection process. All the component profiles are classified under the structures of ATs (see Figure 6). The function matching process just need to compare the user requirements with the component profiles, registered under the selected function classes.

When different software vendors and users in the same domain agree with the same domain model, they come to an agreement about the function of each activity. And there is no semantic difficulty in function matching process. When multi domain models are used in one certain domain, there are semantic problems. In sophistical function matching process, algorithm FCN is used for two ATs' mapping to eliminate the semantic heterogeneity existing in different models. User requirements translation from user AT to component AT is adopted to transform two different models into one.

Even though the mapping table can be generated after the execution of algorithm FCN, it is much time-consuming. Especially, when multi ATs (suppose n ATs) exist in the same domain, $n*(n-1)$ times of mapping should be done to generate the mapping tables between two ATs. In practice, a standard AT can reduce the mapping times to n and facilitate the mapping process (all the ATs just need to map to the standard AT).

There is still much work to be done. In the first scanning of algorithm FCN, a domain synonymy dictionary is used to find the same function node_pairs. When one node can find two and more same function nodes, how to select the fittest one should be serious considered. The complete description with resource and information exchanged model is still need further work.

6. ACKNOWLEDGEMENT

This work is supported by National 863 Plan of China under Grants No. 2006BAF01A48.

The authors thank ISO/TC 184/SC 5/ WG 4 members and its Japanese mirror committee members for their useful effort to international standards on which our work are based and extended.

REFERENCES

Andrea Rodrı´guez M., Egenhofer Max J.. (2003). Determining Semantic Similarity among Entity Classes from Different Ontologies. *IEEE TRANSACTIONS ON KNOWLEDGE AND DATA ENGINEERING*, **Vol. 15**, **NO. 2**, pp. 442-456.

Ayala Claudia, Franch Xavier. (2006). Domain Analysis for Supporting Commercial Off-The-Shelf Components Selection. In: *International Conference on Conceptual Modelling – ER 2006*, pp. 354-370.

Gemma Grau, Juan Pablo Carvallo, Xavier Franch, Carme Quer. (2004). DesCOTS: A Software System for Selecting COTS Components. In: *Proceedings of the 30th EUROMICRO Conference*, pp. 118-126.

Günther Ruhe. (2003). Intelligent Support for Selection of COTS Products. In: *Web Databases and Web Services 2002*, pp. 34–45.

Hung-Ju Chu, Randy Y.C. Chow. (2007). Reaching Semantic Interoperability through Semantic Association of Domain Standards. In: *11th IEEE International Workshop on Future Trends of Distributed Computing Systems, FTDCS '07*, pp. 15-20.

ISO/TC184/SC5/WG4, ISO 16100-1 (2002) *Industrial automation systems and integration – manufacturing capability profiling for interoperability – Part 1: Framework.*

ISO/TC184/SC5/WG4, ISO 16100-2 (2002) *Industrial automation systems and integration – manufacturing capability profiling for interoperability – Part 2: Profiling methodology*.

ISO/TC184/SC5/WG4, ISO 16100-3 (2005) *Industrial automation systems and integration – manufacturing capability profiling for interoperability – Part 3: Interface services, protocols and capability templates*.

Nasib S. Gill. (2003). Reusability Issues in Component-Based Development. *ACM SIGSOFT Software Engineering Notes*, **Vol. 28**, **NO. 4**, pp. 4-4.

SEMATECH Technology Transfer 2706 Montopolis Drive Austin. (1998). *Computer Integrated Manufacturing (CIM) Framework Specification Version 2.0.*

Sofien Khemakhem, Khalil Drira, Mohamed Jmaiel. (2006). SEC-A Search Engine for Component based software development. *SAC'06*, pp. 1745-1750.

Vijayan Sugumaran, Veda C. Storey. (2003). A Semantic-Based Approach to Component Retrieval. *The DATA BASE for Advances in Information Systems*, **Vol. 34**, **No. 3**, pp. 8-24.

W. Yu, M. Matsuda, Q. Wang. (2007). Enhancing Interoperability of Manufacturing Software Units Using Capability Profiling. *Enterprise Interoperability: the new Challenges and Approaches*, Springer, pp. 451-460.

Xavier Franch, Marco Torchiano. (2005). Towards a Reference Framework for COTS-Based Development: A Proposal. ACM SIGSOFT Software Engineering Notes, **Vol. 30, NO. 4**, pp. 1-4.