An Incremental and FCA-based Ontology Construction Method for Semantics-based Component Retrieval

Xin Peng, Wenyun Zhao
Computer Science and Engineering Department, Fudan University, Shanghai, China
{pengxin, wyzhao}@fudan.edu.cn

Abstract

In semantics-based component retrieval ontology is usually employed as the semantic basis for component representation and matching. Existing methods always assume that the ontology already exists. However, ontology construction itself is a big challenge and the construction of ontology for component retrieval has its own characteristics differing from general domain ontology, including the emphasis on functional semantics, the explicit instance set (i.e. semantic descriptions of all the components stored in the repository) and the explicit objective of characterizing components for retrieval.

In this paper, we propose an incremental and FCA (Formal Concept Analysis) based ontology construction method. The feature of incremental ontology construction means a collaborative and interactive process with component providers to collect basic input/output descriptions of components. FCA with the extension of hierarchical-valued concept context is adopted to elicit concepts from original component descriptions to construct ontological hierarchy for functional concepts. After concept analysis, semantic representation of each component can also be filled automatically. So our method can alleviate the efforts of ontology construction remarkably and the quality can also be ensured in terms of the requirements of component retrieval.

1. Introduction

CBSD (Component Based Software Development) is a systematic method for developing software products with reusable components from in-house or commercial component repositories. However, there are still several issues that must be addressed to achieve widespread acceptance of CBSD, including component repositories and easy and widely accepted means of component retrieval with high precision [1]. Traditional keywords and categories based component browse and search have been proved to be insufficient for large-scale component trading and reuse. So, more and more researches have focused on semantics-based component representation and retrieval (e.g. [1-4]). In these methods ontology is usually employed to provide the semantic basis.

These existing methods focus on semantic representation model (e.g. [4]), semantic architecture for distributed repositories (e.g. [3]), refining and matching of user queries (e.g. [1-2]), etc. They always assume that the ontology (usually ontology for specific application domain, called domain ontology) already exists and the construction issue is ignored. In fact, ontology construction itself is a big challenge, especially when the ontology is expected to have relevance and value to a broad audience [5]. In most cases, the ontology is constructed manually by centralized teams, which is a complex, expensive and time-consuming process [6-7]. Accordingly, some collaborative and automatic/semi-automatic methods of ontology construction are proposed. Collaborative methods make the ontology development a joint effort reflecting experiences and viewpoints of persons who intentionally cooperate to produce it [5].

What we focus in this paper is the construction of ontology for component representation and retrieval, which differs from general domain ontology in some aspects. First, functional concepts with domain-specific features are essential, which are the main means for component users to express their requirements. Second, it has the explicit instance set, i.e. semantic descriptions of all the components stored in the repository. Third, it has the explicit objective of characterizing components in the repository so that they can be distinguished for retrieval. So, the ontology should be evaluated by its description capability for the current components in the repository, and can be incrementally constructed along with the continual component submissions.

Aiming at the need of ontology construction for semantic component retrieval and its characteristics, we propose an incremental and FCA-based ontology construction method in this paper. The feature of incremental ontology construction means a collaborative and interactive process with component providers to collect basic input/output descriptions of components. FCA with the extension of hierarchical-valued concept context is adopted to elicit concepts

Seventh International Conference on Quality Software (QSIC 2007)
0-7695-3035-4/07 $25.00 © 2007 IEEE

from original component descriptions to construct ontological hierarchy for functional concepts. After concept analysis, semantic representation of each component can also be filled automatically. So our method can alleviate the efforts of ontology construction remarkably and quality of the ontology and component representation can also be ensured.

The remainder of the paper is organized as follows: Section 2 introduces some backgrounds and related works, and states features of our method. Semantic representation for components and overview of our ontology construction process are provided in section 3. Section 4 presents the hierarchical-valued FCA algorithm for concept hierarchy generation. Section 5 introduces the conversational process with providers. A case study in the ECommerce domain is given in section 6. Finally section 7 concludes the paper.

2. Backgrounds and related works

2.1. Collaborative ontology construction

Ontological commitment is the agreement by multiple parties to adopt a particular ontology for specific domain of interest [5]. It is essential for the ontology to be well understood and accepted. Working toward ontological commitment should not be an afterthought, but rather an integral aspect of ontological engineering, which underlies the collaborative approach to ontology design [5].

Clyde et al. [5] propose a collaborative approach to ontology design involving a sizable, balanced mix of experienced scholars and practitioners. Their method begins with an initial ontology, which seed the collaborative design activity and serves as an anchor to help focus the attention of collaborators. Benjamin et al. [7] report their collaborative ontology construction work for the life sciences carried out in a health conference. A web-accessible, template-based interface is used to facilitate collaborative ontology development.

2.2. Ontology construction by FCA

Formal concept analysis (FCA) is a theory of data analysis which identifies conceptual structures among data sets [8]. The basic structure of FCA is the context, which is comprised of a set of objects, a set of attributes and relations describing which objects possess which attributes [6]. Here we introduce the definitions of context, concept and concept order given in [9], similar definitions can also be found in other literatures on FCA.

Definition 1. (Formal context) A formal context is a triple $<G, M, I>$. G is a set of objects, M is a set of descriptors, and I is a binary relation that $I \subseteq G \times M$.

The notation $<y, x> \in I$ denotes that an object $y \in G$ has an attribute or descriptor $x \in M$. For example, "*sparrow*" is an object with the descriptor "*can fly*".

Definition 2. (Formal concept) A formal concept is a pair of sets $<Y, X>$, where $Y \subseteq G$ and $X \subseteq M$. Each pair must be complete with respect to I, which means that $Y'=X$ and $X'= Y$, where $Y'=\{x \in M | \forall y \in Y, <y, x> \in I\}$ and $X'=\{y \in G | \forall x \in X, <y, x> \in I\}$.

The set of descriptors of a formal concept is called its intent, while the set of objects of a formal concept is called its extent. The correspondence between intent and extent of complete concepts is a *Galois* connection between the power set $P(M)$ of the set of descriptors and the power set $P(G)$ of the set of objects [6][9]. The *Galois* lattice L for the binary relation is the set of all complete pairs of intents and extents, with the following partial order.

Definition 3. (Concept order) Given two concepts $N_1=<Y_1, X_1>$ and $N_2=<Y_2, X_2>$, $N_1 < N_2 \leftrightarrow X_1 \supseteq X_2$. The dual nature of the *Galois* connection means we have the equivalent relationship $N_1 < N_2 \leftrightarrow Y_1 \subseteq Y_2$.

Nowadays, FCA has also been widely employed in ontology construction (e.g. [6][8-9]). In these works, the concept lattice produced by FCA, which factors out commonalties while preserving specialization relationships between concepts [6], is regarded as the ontology hierarchy.

2.3. Features of our method

Different from the works introduced above, we focus on the construction of domain ontology with specific intention of semantic component retrieval. On the one hand the ontology must meet the requirements of component representation and matching. On the other hand, the ontology can be incrementally built with the continual component submissions. Comparing with other FCA-based methods (e.g. [6][8-9]), our method has two characteristics. First, the formal context (i.e. input/output semantics of components in our method) is collected through system-dominant conversations with component providers. Second, the FCA algorithm is extended to support conceptual context with hierarchical values.

3. Component representation and ontology construction process

Function is the primary point of interest when determining reusability [10]. So, in our method component semantics and ontology concepts are functional semantics specifying business operations.

3.1. Component Representation

In our previously proposed ontology-based feature model, operations with business semantics are

Table1. Intent/extent and hyponymy relations of generated *Action* concepts

Action	Intent				Extent (Component)	Father *Action*
	Input Semantics		Output Semantics			
	Object	Facet Value	*Object*	Facet Value		
PayByUnionPay	*Bill*	*hasState=ToPay*	*Bill*	*hasState=Paid*	1	*PayByBankCard*
	BankCard	*issuedBy=UnionPay*	*Receipt*			
PayByVisa	*Bill*	*hasState=ToPay*	*Bill*	*hasState=Paid*	2	*PayByBankCard*
	CreditCard	*issuedBy=Visa*	*Receipt*			
PayByBankCard	*Bill*	*hasState=ToPay*	*Bill*	*hasState=Paid*	1, 2	*OnlinePayment*
	BankCard		*Receipt*			
PayByPaypal	*Bill*	*hasState=ToPay*	*Bill*	*hasState=Paid*	3	*OnlinePayment*
	PayPalAccount		*Receipt*			
OnlinePayment	*Bill*	*hasState=ToPay*	*Bill*	*hasState=Paid*	1, 2, 3	*Payment*
			Receipt			
OfflinePayment	*Bill*	*hasState=ToPay*	*Bill*	*hasState=Pending*	4	*Payment*
			Receipt			
Payment	*Bill*	*hasState=ToPay*	*Receipt*		1, 2, 3, 4	*BillProcessing*
PaymentConfirm	*Bill*	*hasState=Pending*	*Bill*	*hasState=Confirmed*	5	*BillProcessing*
BillProcessing	*Bill*		*Bill*		1, 2, 3, 4, 5	

fundamental elements [11]. So we distill functional semantics related elements from the ontology-based model and extend the notation of business objects to compose the ontology meta-model for component representation, which is presented in figure 1.

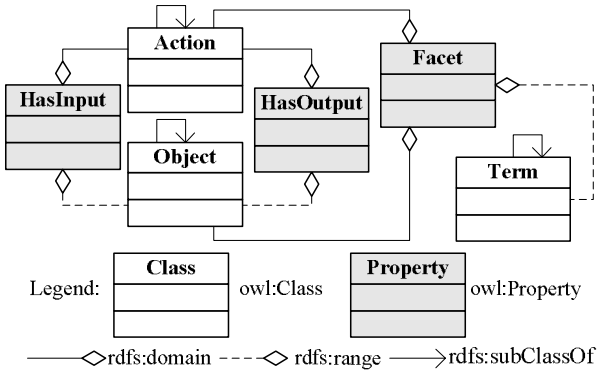

Figure 1. Ontology meta-model for Component Representation

In figure 1, concepts are represented by OWL [12] elements. Three OWL classes (*Action*, *Object*, *Term*) and three OWL properties (*HasInput*, *HasOutput*, *Facet*) are defined. Domain and range of OWL properties are represented by aggregation elements with solid and dashed lines respectively. *Action* represents business operations, which has specific business objects as its input and output. So, two properties of *HasInput*, *HasOutput* are defined between *Action* and *Object*. In order to provide more business semantics, we introduce the concept of *Facet* as dimension of precise description for *Action* and *Object*. Restricted value space for *Facet* is represented by *Term* [11].

Simplified expressions for preconditions and effects are also involved in our input/output semantics, which are denoted by facet descriptions of input/output objects. For example, in table 1, *PaymentConfirm* has output *Bill* with the facet value "*hasState=Confirmed*", which means after *PaymentConfirm* the bill will be confirmed to be paid or failed. A component can have multiple inputs or outputs, and the output can be seen as effect of the operation besides real output, such as the output of *PaymentConfirm*. In our method, each *Action* concept denotes a business operation with specific input/output semantics, while facets on *Action* concepts depict different features may implemented by components with the same input/output semantics.

A segment of the ECommerce domain ontology is demonstrated in figure 2, which are divided into two parts. The segment under the line is assumed to be preexisting, depicting all the business objects, which can be acquired from traditional domain ontology. Above the line is *Action* concept hierarchy and related facet descriptions, which are constructed by our method. The root *Action* concept is *BillProcessing* with two sub-concepts of *Payment* and *PaymentConfirm*. In our ontology model, intents of functional concepts are input/output semantics, as showed in table 1, extents in which refer to the components in the case study of section 6. We can see the concept hierarchy is constructed on their intents. For example, *PayByUnionPay*, which accepts all the UnionPay [1] issued bank cards for payment, and *PayByVisa*, which accepts Visa credit cards, are both sub-concepts of *PayBybankCard* in that they have the common input semantics of to-pay *Bill* and *BankCard* and common output semantics of paid *Bill* and *receipt*. Facets characterize additional features of components which

[1] China UnionPay is an association for China's banking card industry, which provides payment gateway for UnionPay bank cards.

0-7695-3035-4/07 $25.00

component users concern, e.g. *hasLogOutput* denotes the logging policy (*NoLog*, *LogFile* or *Console*) beyond the basic input/output semantics.

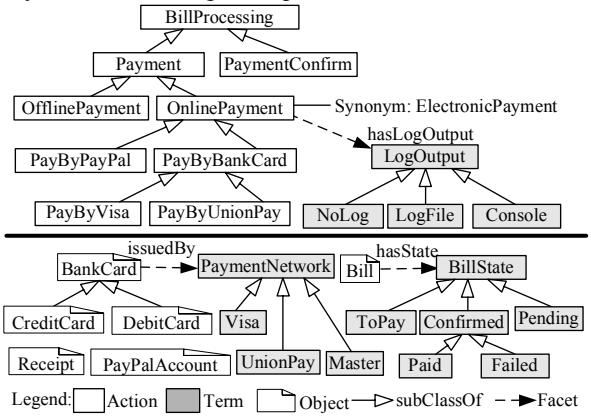

Figure 2. Segment of the ECommerce domain ontology

Now we can define the semantic representation of component functions on the ontology meta-model.

Definition 4 (Facet set and facet value): For $\forall c \in Action$ *or* $c \in Object$, *FacetSet(c)* denotes the set of all the effective facets for *c*. The notation fv_c represents the original facet-value function of *c* in the ontology and can be stated as $fv_c=[f_1=v_1, f_2=v_2, \ldots f_n=v_n]$ ($f_i \in FacetSet(c)$, $v_i=fv_c(f_i)$, $1 \le i \le n$).

Definition 5 (Concept semantics and semantics refinement): An *Action* or *Object* semantics is a pair $s=<c, fv>$, in which $c \in Action$ or $c \in Object$ and fv: $FacetSet(c) \rightarrow Term$ is the facet-value functions of *s* on all the facets defined on *c*. For two *Action* or *Object* semantics $s_1=<c_1, fv_1>$ and $s_2=<c_2, fv_2>$, we say s_2 is the refinement of s_1 if it is true that (c_2 *rdfs:subClassOf* c_1)□($\forall f \in FacetSet(c_1).(fv_2(f)$ *rdfs:subClassOf* $fv_1(f)$)). The relation can be represented as $s_2 \le s_1$.

Definition 6 (Functional semantics of component): The functional semantics of a component *comp* is a pair $semantics(comp)=<a, fv>$, in which $a \in Action$, it is the business operation implemented by *comp*, and *fv* is the facet-value function of *comp*. For each component *comp* with $semantics(comp)=<a, fv>$, it is true that $semantics(comp) \le <a, fv_a>$.

For example, a component with the semantics <*PayByVisa*, [*hasLogOutput=LogFile*]> means it is a payment component for Visa cards with logging file.

3.2. Process of Ontology Construction

The process of collaborative ontology design could range from being strongly anchored with a proposed ontology as the starting point, to comparatively unstructured serendipitous discussion [5]. An anchoring ontology can apparently improve quality of the collaboratively constructed ontology. Our method focuses on acquisition of business functions related concepts and relations, while concept semantics of business objects is assumed to exist already, which are the main theme of traditional domain ontology.

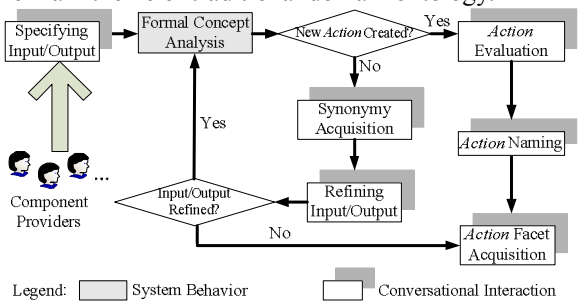

Figure 3. Process of the incremental ontology construction

Figure 3 depicts the process of our incremental ontology construction. Component providers are participators of the collaborative ontology construction. They specify input/output semantics when submitting a component to the repository, including the business objects and their facet descriptions. Then incremental formal concept analysis is performed on the new object [2], i.e. input/output descriptions of the new component. After that, there may be two possibilities:

New *Action* concepts created. That means the component implements a new business function. An *Action* directly corresponding to the component and several possible ancestor concepts may be created. Not all of the generated ancestor concepts are meaningful in the domain, so the provider is prompted with generated concepts for evaluation. Then meaningless concepts are omitted, and the provider is requested to name the reserved new concepts, each with the prompt of input and output semantics. After that, functional concept of the component semantics is filled with the most refined one of the newly created *Action* concepts.

No *Action* concepts created. That means the component has the same input/output semantics with an existing *Action* concept. Then the provider is requested to contribute synonymies for related concepts. Duplicate *Action* concepts may mean rough input/output descriptions, so the system will prompt the provider to refine current input/output descriptions. If more refined descriptions are given, another cycle of concept analysis will startup. Otherwise, the existing *Action* concept will be assigned to the component.

Now the submitted component has been assigned with a newly created or existing *Action* concept according to the input/output semantics. Then the system will interact with the provider to acquire facet descriptions for the component, since components with

[2] The "object" here refers to the object in formal context defined in Definition 1.

the same input/output semantics may differ in implementation features.

4. Concept hierarchy generation by FCA

In classic FCA descriptors are properties that an object may have or not, called one-valued attributes. However, in our method the input/output descriptions are attributes for concept analysis, which have hierarchical value space (i.e. business object concepts), so we extend FCA further to support hierarchical-valued attributes. We will redefine the formal context and formal concept according to the concept and component semantics defined in 3.1.

Definition 7. (Formal component context) A formal component context is a quadruple $<G, M, I, O>$, in which G is a set of components, $M=\{HasInput, HasOutput\}$, O is *Object* semantics, and I is the input/output descriptions such that $I \subseteq G \times M \times O$.

The notation $<g, m, o> \in I$ is used to express the fact that a component g ($g \in G$) has an input/output (indicated by m) semantics $o \in O$. For example, for the first component $comp_1$ in table 2, there is $<comp_1, HasInput, <Bill, [hasState=ToPay]>> \in I$.

Definition 8. (Formal *Action* concept) A formal *Action* concept is a pair $<Y, X>$, where $Y \subseteq G$ and $X \subseteq M \times O$. Each pair must be complete with respect to I, which means that $X'=Y$ and $Y'=X$, where $X'=\{g \in G | \forall (m, o) \in X, \exists <g, m, o'> \in I \wedge o' \leq o\}$ and $Y'=\{(m, o) \in Y'' | \neg \exists (m, o') \in Y'', o' \leq o \wedge o' \neq o\}$, in which $Y''=\{(m, o) \in M \times O | \forall y \in Y, \exists <y, m, o'> \in I \wedge o' \leq o\}$.

We can see that formal *Action* concept reserves the essence of formal concept, i.e. a pair of intent and extent mapped into each other by the *Galois* connection [6]. The difference is that intent of an *Action* concept is not exactly the same properties owned by its extent components, but common input/output semantics abstracted from their semantics. Then we can give the algorithm of concepts and hyponymy relations generation by FCA as follows.

Algorithm 1. Concepts and Hyponymy Relations Generation

```
// INPUT : a formal context <G, M, I, O>
// OUTPUT : Action Concept Set CSet and hyponymy relations
HRel
for all g∈G do
   CSet←CSet∪{(extent(intent(g)), intent(g))};
end for
for all c∈CSet do
   for all g∈(G−extent(c)) do
      X←extent(c)∪{g};
      if intent(X)≠φ and (extent(intent(X)), intent(X))∉CSet then
         CSet←CSet∪(extent(intent(X)), intent(X));
      end if
   end for
end for
for all c1∈CSet do
   for all c2∈CSet−{c2} do
      if (c1< c2)∧(¬∃c3∈CSet−{c1,c2}[(c1< c3)∧(c3< c2)]) then
   HRel←HRel∪{(c1, c2)}
      end if
   end for
end for
```

Extent computation is comparatively easy by checking each component with the extent definition X' given in Definition 8, so we only present the algorithm of intent computation here.

Algorithm 2. Intent Computation

```
// INPUT : a formal context <G, M, I, O> and component set Y⊆G
// OUTPUT : intent(Y), the intent set of Y according to Definition 8
intent(Y)←φ;
for all y∈Y do
   if intent(Y)=φ then
      intent(Y)←{(m, o)∈M×O|<g, m, o>∈I};
   else
      for all (m, o)∈intent(Y) do
       accepted←false; // whether (m, o) can be accepted by y
       for all o'∈{o'∈O|<g, m, o'>∈I} do
        x←o∪o'; // the union of o and o'
            if x≠NULL then // (m, o) is accepted by y
          intent(Y)←intent(Y)∪{(m, x)}−{(m, o)};
          accepted←true;
              break;
         end if
       end for
       if accepted=false then
        intent(Y)←intent(Y)−{(m, o)}; //remove from the intent
       end if
     end for
   end if
end for
```

In the algorithm, operator "∪" denotes the concept union of two business object semantics according to the following definition.

Definition 9 (Union of object semantics): For two *Object* semantics $s_1=<o_1, fv_1>$ and $s_2=<o_2, fv_2>$, if there exists $s=<o, fv>$ satisfies $o \neq Object \wedge s_1 \leq s \wedge s_2 \leq s \wedge (\neg \exists s' \neq s, s_1 \leq s' \wedge s_2 \leq s' \wedge s' \leq s)$, then there is $s_1 \cup s_2 = s$, otherwise $s_1 \cup s_2 = NULL$, which means no consentience can be reached on their concept semantics.

The FCA process is conducted in each component submission, so the execution efficiency is essential. In our method, formal context of each FCA execution can be confined to those components having at least one common input or output semantic *Object* with the newly submitted component. For example, in table 1, *PayByPayPal* and *PaymentConfirm* have the common input *Bill*. After each FCA process, if components with exactly the same input/output semantics with the new component do not exist, then new *Action* concepts will be generated, including an exact *Action* corresponding to the new component and several possible abstract *Action* concepts. These new abstract concepts will be evaluated by the provider and meaningless concept will be discarded, as showed in figure 3.

5. Semantics-based interactions

5.1. Input/output acquisition

In the specification of input/output semantics, elicitation of *Object* concept is the main challenge, since having a scale of hundreds and thousands *Object* concepts is natural in a not so small business domain while facets defined on an *Object* concept are not so many. Elicitation of *Object* concept can be seen as the issue of mapping provider expressions to existing ontology concepts. In our method, providers are assumed to specify the input/output information by nominal phrases with possible attribute modifiers, such as "Visa card" or "pending bill". Then the mapping schema can be established as: nouns->*Object* concepts and attributes->facet values. Several similarities can be adopted here to contribute the concept matching, such as lexical similarity and contextual similarity. Due to the limitation of space, detailed description of concept mapping is omitted here. The interactive process of input/output specification is as follows:

1. Prompt the provider to add an input or output description for the submitted component;
2. Try to map provider specified phrase to an existing *Object* concept by lexical and contextual similarity;
3. If an obvious concept mapping can not be reached or the mapping is denied by the provider, prompt the provider to change the expression or choose from several possible candidates, and then repeat 2;
4. Guide the provider to refine the mapped *Object* concept by exploring along the hierarchical relations of *Object* concepts;
5. Let *o* be the final determined *Object* concept, guide the provider to fill the facet value $fv(f)$ for each $f \in FacetSet(o)$;
6. Construct an *Object* semantics $<o, fv>$ as an input or output semantics of the submitted component;
7. Repeat 1-6 until no more input/output description can be added.

In each concept analysis, if no new *Action* concepts generated, then its input/output semantics is duplicate with that of an existing component, maybe due to rough descriptions. So the provider will be requested to refine current input/output descriptions. The process is similar to the input/output specification, but will provide detailed information (full textual descriptions and user feedbacks) of duplicate components to help the provider to distinguish among them.

5.2. *Action* evaluation and naming

In our method, *Action* concepts are generated from input/output semantics of components by FCA. In FCA, concepts are automatically derived from the more or less commonalities contained in a set of components, so not all generated concepts can be accepted as a meaningful business concepts. Consequently each newly generated abstract *Action* concept will be evaluated by the provider. If the concept is meaningful, the provider is also requested to name it from his comprehension. Names for the same concept can be different for different providers, which is the problem of synonymy and can be resolved by synonymy acquisition discussed in the next subsection.

5.3. Synonymy and *Action* facet acquisition

If a submitted component has exactly the same input/output semantics with an existing component, no new *Action* concepts will be generated. Developing and submitting a component with reduplicate function usually means the provider is familiar with related business concepts, so the provider will be consulted for possible synonymies for existing *Action* concepts. In synonymy acquisition, the reduplicate *Action* concept and all the ancestor concepts of it will be chosen. According to figure 2, for example, if a new *PayByPayPal* component is submitted, the following concepts will be taken out to acquire synonymies: *PayByPayPal*, *OnlinePayment*, *Payment*, and *BillProcessing*. The provider will be prompted with the question like: *Do you know any other term or phrase with the same meaning of "OnlinePayment"*.

Whether new *Action* concepts are created or not, facet descriptions are requested from the provider at the end of each component submission. *Action* facet acquisition is to guide the provider to further characterize his component not by new *Action* concept, but by additional semantic facets on existing *Action* concept. The provider can assign the value for an existing facet, or propose a new facet to add a descriptive dimension to distinguish his component from other similar components. In facet value specification, the system can list all the candidate terms in a hierarchical mode for each facet of the determined *Action* concept. The provider can choose a term for his component. He can also create a new facet for corresponding *Action* concept by assigning the name and available terms for the facet.

6. Case study

This section demonstrates the incremental and FCA-based construction process of the ontology segment showed in figure 2. Table 2 lists five submitted components with their input/output semantics by the order of submission. We take the contrast between FCA on the first three components (1-3) and all the five components (1-5) to demonstrate the incremental construction. Figure 4 depicts the contrast effect, and intents (input/output semantics), extents (components) and hyponymy relations of all the generated *Action* concepts are listed in table 1. Along with the continual component submission, more and

more components with the same basic function of *OnlinePayment* may be submitted, so providers are requested to provide more features for their components. For example, a provider may propose the policy of logging as an outstanding characteristic of his component, so he creates the facet *hasLogOutput* for *OnlinePayment* with options of *NoLog*, *LogFile*, and *Console*.

Table2. Input/output semantics of submitted components

ID	Input Semantics		Output Semantics	
	Object	**Facet Value**	***Object***	**Facet Value**
1	*Bill*	*hasState=ToPay*	*Bill*	*hasState=Paid*
	BankCard	*issuedBy=UnionPay*	*Receipt*	
2	*Bill*	*hasState=ToPay*	*Bill*	*hasState=Paid*
	CreditCard	*issuedBy=Visa*	*Receipt*	
3	*Bill*	*hasState=ToPay*	*Bill*	*hasState=Paid*
	PayPal -Account		*Receipt*	
4	*Bill*	*hasState=ToPay*	*Bill*	*hasState=Pending*
			Receipt	
5	*Bill*	*hasState=Pending*	*Bill*	*hasState=Confirmed*

7. Conclusion and future work

In this paper, we propose an incremental and FCA-based ontology construction method for semantics-based component retrieval. In our method, the ontology is constructed on the contribution of numerous component providers, and the concept structure is generated automatically, so the ontology can be avoided to be one-sided. In order to eliminate conflicts in the collaboration, ontology segment on business objects of the domain is assumed to preexist as the anchoring ontology for input/output semantics. After concept analysis, semantic representation of each component can also be filled automatically. So our method can alleviate the efforts of ontology construction remarkably and the quality can be ensured according to the requirements of component retrieval.

The functional ontology concepts generated by our method is adopted to represent component semantics and user quires. In other works, we also research on user query generation and semantic matching on functional ontology concepts. In our future research, we will integrate these works to establish a system of methods and tools for semantic component retrieval.

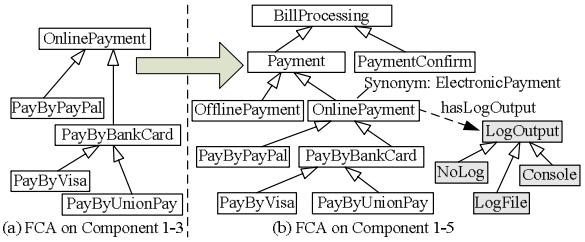

Figure 4. The result of incremental ontology construction

Acknowledgments. This work is supported by the National Natural Science Foundation of China under Grant No. 60473061 and No.60473062.

8. References

[1] Vijayan Sugumaran, Veda C. Storey. A Semantic-Based Approach to Component Retrieval. ACM SIGMIS Database, Vol. 34, No. 3, 2003.

[2] Haining Yao, Letha Etzkorn. Towards A Semantic-based Approach for Software Reusable Component Classification and Retrieval. The 42nd annual Southeast regional conference, 2004, ACM Press.

[3] Regina M. M. Braga, Marta Mattoso and Cláudia M. L. Werner. The Use of Mediation and Ontology Technologies for Software Component Information Retrieval. SSR'01, 2001, ACM Press.

[4] Cristiane A. Yaguinuma, Marilde T. P. Santos, and Marina T. P. Vieira. Ontology-Based Meta-model for Storage and Retrieval of Software Components. Proceedings of the 31st VLDB Conference.

[5] Clyde W. Holsapple, K. D. Joshi. A Collaborative Approach to Ontology Design. Communications of the ACM, Vol. 45, No. 2, 2002.

[6] Suk-Hyung Hwang, Hong-Gee Kim, and Hae-Sool Yang. A FCA-Based Ontology Construction for the Design of Class Hierarchy. ICCSA 2005, Springer LNCS 3482, 2005.

[7] Benjamin M. Good, Erin M. Tranfield and Poh C. Tan, etc. Fast, Cheap and Out of Control: A Zero Curation Model for Ontology Development. Pacific Symposium on Biocomputing, 2006.

[8] Marek Obitko, Vaclav Snasel, and Jan Smid. Ontology Design with Formal Concept Analysis. CLA2004.

[9] Michael Bain. Inductive Construction of Ontologies from Formal Concept Analysis. AI 2003, Springer LNAI 2903.

[10] John Penix, Phillip Baraona, Perry Alexander. Classification and Retrieval of Reusable Components Using Semantic Features. KBSE 1995: 131-138.

[11] Xin Peng, Wenyun Zhao, Yunjiao Xue, Yijian Wu. Ontology-Based Feature Modeling and Application-Oriented Tailoring. Proceedings of the 9th International Conference on Soft-ware Reuse (ICSR2006), Springer LNCS 4039.

[12] Sean Bechhofer, et al. Owl Web Ontology Language Reference", http://www.w3.org/TR/owl-ref/, 2004-02-10.