Formal Ontology: Foundation of Domain Knowledge Sharing and Reusing

LU Ruqian (陆汝钤)[1,2,3] and JIN Zhi (金 芝)[1]

[1] *Key Laboratory of Management, Decision and Information Systems*
Institute of Mathematics, Academy of Mathematics and System Sciences
Key Laboratory of Intelligent Information Processing, Institute of Computing Technology
The Chinese Academy of Sciences, Beijing 100080, P.R. China

[2] *Open Laboratory of Intelligent Information Processing, Fudan University, Shanghai 200433, P.R. China*

[3] *Key Laboratory of Multimedia and Intelligent Software, Beijing Polytechnic University*
Beijing 100044, P.R. China

E-mail: rqlu@math08.math.ac.cn; zhijin@amss.ac.cn

Received January 14, 2002; revised May 10, 2002.

Abstract Domain analysis is the activity of identifying and representing the relevant information in a domain, so that the information can be shared and reused in similar systems. But until now, no efficient approaches are available for capturing and representing the results of domain analysis and then for sharing and reusing the domain knowledge. This paper proposes an ontology-oriented approach for formalizing the domain models. The architecture for the multiple-layer structure of the domain knowledge base is also discussed. And finally, some genetic algorithm-based methods have been given for supporting the knowledge sharing and reusing.

Keywords domain knowledge model, virtual domain model, knowledge sharing and reusing, formal ontology, genetic algorithm

1 Introduction

As it was pointed out by Maymir-Ducharme in [1] that "we can no longer treat each new project as a single, new and independent development effort and not build on previous engineering efforts and experience", software belonging to the same application domain has lots of similarity. This similarity comes from the expert knowledge of the application domain. Therefore, "instead, we need to view these systems within the context of similar systems built in the past, exploring the commonalties and engineering the appropriate variances". Better we do not treat the endless software development as a series of separate projects, but treat the application domain as a whole. Then, we can first make a thorough analysis of the domain characteristics and then make reuse of the knowledge and experiences gained from this analysis whenever we are going to develop new software in this domain. People call this idea *domain analysis*.

In the context of software reuse, the concept of *domain analysis* was introduced by J. Neighbors in his Ph.D. thesis[2] to refer to "the activity of identifying the objects and operations of a class of similar systems in a particular problem domain."[3] He further pointed out that "Domain analysis is the process of identifying, collecting, organizing, and representing the relevant information in a domain, based upon the study of existing systems and their development histories, knowledge captured from domain experts, underlying theory, and emerging technology within a domain. The product of domain

Partly supported by the National Natural Science Foundation of China (Grant No.69983010), the National Natural Science Key Foundation of China (Grant No.69733020), the National High-Tech Project (Grant No.2001AA113130), the Pre-973 project (Grant No.2001CCA03000) and the knowledge innovation program of the Chinese Academy of Sciences.

analysis is domain model." Fig.1 summarizes the inputs, outputs and agents that support the domain analysis process.

More than 10 years passed since the concept of domain analysis was proposed. Also, many models of domain analysis have been proposed. For example, the IEEE standard ODM[5] (Organization Domain Modelling) is a delicate and well designed method. The FODA[6] (Feature Oriented Domain Analysis) of SEI (Software Engineering Institute) is one of the most well-known methods, which produces domain information model, domain characteristics model and domain operation model. The method of Prieto-Diaz divides the domain analysis into three phases: pre-domain analysis, domain analysis and post-domain analysis[4]. The method of McCain, on the other hand, gives a two phase definition: the high level domain analysis and the component domain analysis[7]. The JB-OODA[8] (Jade Bird Object-Oriented Domain Analysis, a part of the Jade Bird project[9]) method proposed a three-stage procedure: background modelling, domain modelling and architecture modelling. This approach roughly corresponds to the three phases of the FODA approach, but the modelling techniques are different from it.

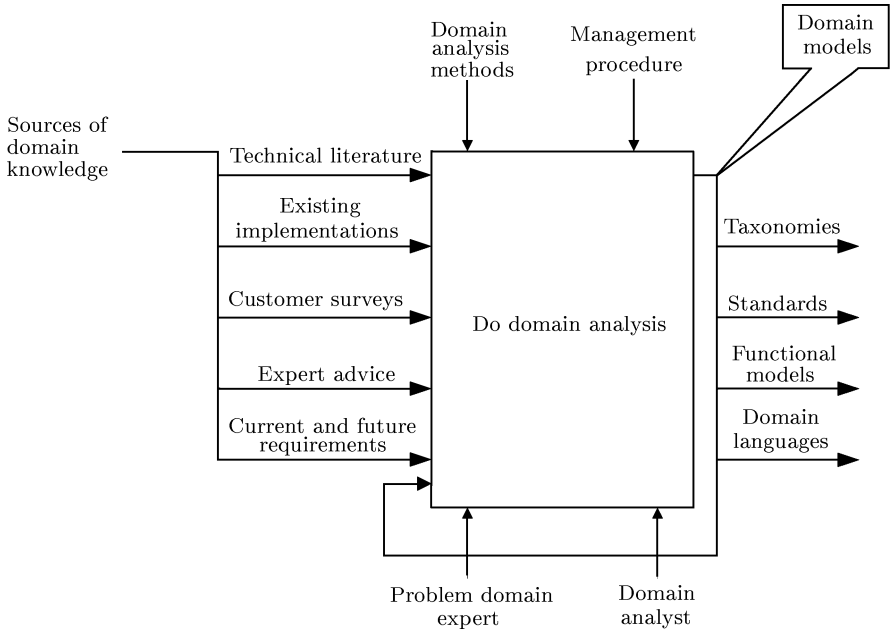

Fig.1. Domain analysis process[4].

Nowadays, domain analysis is used successfully in several different application domains, such as the air traffic control, the flight dynamics, the public telephone exchanges and so on. These efforts focus on identifying, classifying and storing individual reusable components, such as data and process components and object classes. However, there are still no systematical methods for domain analysis which can be followed step by step to build the complete domain models, specially the domain requirements models or can be used to build the domain models automatically.

Since the 90's, a more general concept, *domain engineering*, has been proposed. "The term domain engineering is used to describe the technology (concepts, methods, processes and tools) developed to extend the traditional single system engineering paradigm (referred to as application engineering) to consider multiple (common or related) systems in order to engineer systems faster, better and cheaper"[1].

According to the point of view of some researchers, even the concept *domain engineering* is only one of the several levels of techniques for engineering software which are different in their levels of abstraction and scope. There should be three levels: the enterprise engineering, the domain engineering and

the application engineering. The application engineering is concerned with a single system/application, whereas the domain engineering takes into account multiple similar or related systems, and the enterprise engineering looks at an entire enterprise/organization's high level data and operational needs. Each of these three levels consists of three spaces, i.e., the problem space (requirements), the solution space (design) and the implementation space (software). The relations between them are shown in Table 1.

Table 1. Three Levels of Software Engineering Process

	Problem Space	Solution Space	Implementation Space	Products
Enterprise Engineering	Enterprise Analysis	Enterprise Design	Enterprise Implementation	Enterprise Model Enterprise Architecture
Domain Engineering	Domain Analysis	Domain Design	Domain Implementation	Domain Model Domain Architecture, Domain Assets
Application Engineering	Requirements Analysis	Application Design	Implementation	Requirement Specification Design Specification Software

R. Capilla proposed another division of domain engineering levels[1]. Though the number of levels is also three, he considers it from the point of view of knowledge reuse. The proposed levels include the super-domains, the semantic domains and the specialization domains. In fact, his super-domains contain domain independent knowledge of the knowledge based software construction in the conventional sense. His semantic domains contain knowledge of different application domains and include a set of generic problem statements. So we believe his idea is deeply influenced by the work of Chandrasekaran on generic tasks[11]. His specialization domains are evolved from the semantic domains with more specific knowledge of individual systems. It corresponds to the application engineering of [1].

Since the early 90's, we have been engaged in a national key project, i.e., the Eagle project[12], which is a part of the Jade Bird project. In this project, we have taken a knowledge-based approach. This approach is aiming at automatically generating information systems. The process of system generation should be fully automatic and the integration of subsequent steps should be seamless, from understanding the domain customers till generating the final program.

The feasibility of this approach is based on the premise that the knowledge of both the domain experts and the software engineers can be captured, represented and reused to support the development of information systems. That is why we call this approach KISSME (Knowledge Intensive Software System Manufacture Engineering)[13]. Different from the other available efforts on knowledge-based software engineering, we have paid special attention to the representation and reuse of domain knowledge. The use of domain knowledge is especially important in the automation of the early stages of our information system development, including domain customers' requirement determination, specification design, conceptual modelling and architecture planning. Based on KISSME approach, we have developed a tool called PROMIS (PROtotyping Management Information Systems)[14−19]. With sufficient domain knowledge and necessary business information from customers, PROMIS can automate a large part of application development.

Centered in KISSME approach is the ontology-oriented domain analysis and modelling approach. This approach is somewhat different from those approaches mentioned above. In our approach, there are only two levels: the domain engineering and the application engineering. Our second level coincides with their third level. But our first level includes their first level (the enterprise engineering) and second level (the domain engineering), and still more. Namely, within domain engineering, we are not limited to two or three levels. We allow the domain experts to construct new domain models based on higher-level domain models. The number of levels is not limited.

This paper focuses on the formal representation and organization of the ontology-oriented knowledge base in KISSME approach. It is organized as follows. Section 2 states our basic thought about using ontology. Section 3 presents a mathematic model of ontology. The architecture of the ontology-

oriented knowledge base is given in Section 4. Section 5 proposes a genetic algorithm-based method for knowledge sharing and reusing. And finally at Section 6, we get some conclusions.

2 Using Ontology

2.1 Need for Ontology

Originally, ontology was a concept in the philosophy. It describes the essence and composition of the world, as said by the philosophers. In computer science, ontology is mainly used for knowledge representation. People who are involved in ontology researches are very different. They are distributed in the fields of artificial intelligence and software engineering, philosophy and linguistics, and so on.

There are several well-known examples of approaches for ontology development. The Enterprise Ontology[20] developed by Artificial Intelligence Application Institute in Edinburgh University within the Enterprise Project is a collection of terms and definitions for providing a framework of enterprise modelling. The second example is the TOVE[21] (TOronto Virtual Enterprise) ontology set of the Toronto University in the field of business processes and activities modelling.

A phenomenon we have noticed is that the major part of papers on ontology are first published in journals or conferences which are related to artificial intelligence and knowledge engineering. This fact should not mean that ontology is only interesting for the AI researchers. In its essence, ontology provides a means for knowledge sharing which is very much desirable in large-scale knowledge based projects, including the traditional software engineering projects. Nowadays, we noticed also that more and more software engineers became interested in studying and using ontology concepts in their research projects.

The use of ontology in software engineering has also its own roots. Remember that the entity relationship diagram[22] of Chen can be viewed as one of the simple versions of ontology representation. Also the conceptual graphs[23] remind us of lots of characteristics of modern ontology researches.

In 1996, IEEE has published a standard, i.e., the IEEE standard 1074–1095[24], for developing software life cycle processes. Later, Fernandez Lopez has analyzed the development methodology of several ontology researches and compared them with this standard[25]. From this analysis, we have discovered a crucial point that prevents us from making direct benefit from the results of the current researches on ontology. That is, almost no ontology has been designed for their use in automated machine processing. They are mainly used for communication between human beings. Clearly, such kind of ontologies cannot be used to automate the software production procedure.

2.2 Points for Using Ontology

Our goal is to develop a methodology of ontology-oriented domain analysis in the field of information system modelling. Formal ontology is the prerequisite for modelling domains to support domain knowledge sharing and reusing. However, before presenting this formalized definition, we should illustrate some of our main ideas as the background of this formalization. This background will help us to understand the rather abstract notations in the formalized definition.

Idea 1 (Make Relations as Independent Knowledge Units). In a pure object-oriented paradigm, the multilateral relations are usually attached to this or that object without enough justification. In our opinion, they are attached in a less or more arbitrary way. In our approach, two generalizations are made with respect to the relation concept. First, with exception of the inheritance relation, i.e., the ancestor-offspring relation, all relations are listed explicitly outside of the objects. Objects and relations are now elements of equal importance in an ontology, just as nodes and arcs are of equal importance in a graph. This point makes our paradigm different from the pure object-oriented paradigm which only emphasizes to take objects as basic system elements and considers the relations between objects only as having second order importance. Second, in our paradigm, the relations may have any finite arity, i.e., a multilateral relation having any finite number of parameters. For example

'*human uses a computer to write a text*' could be represented as a multilateral relation '*text-writing* (*writer, tool, written-text*)'.

Idea 2 (Organize the Objects in Ontologies). A single relation may be not enough to characterize the relationship among many objects. In a society, for example, there are family relations, relative relations, friend relations, employer-employee relations, teacher-student relations, etc. All these relations together form our society. It is natural to consider the whole set of these relations together with the objects they connect as an ontology, the society ontology.

Grouping the objects in ontologies has another advantage. Remember that increasing grain size of object representation has been one of the basic techniques in object-oriented analysis used to lower down the complexity of large sized software projects. The concept of object clustering is considered as one of the basic methods to enlarge the object grain size. But a simple merging of objects without a rigorous definition of functional relations and interfaces among these objects may lead to chaos in the object community. This is another reason why we need the concept of ontology.

Idea 3 (Take Objects as Basic Elements of an Ontology). This point makes our approach different from those ontology studies reported in literature, where the researchers take concepts as basic elements of an ontology. In most cases these concepts are considered as atoms and there is no further explanation about the structure of these concepts. Our study about the use of ontology in information systems modelling has shown that a pure ontology-oriented paradigm with structureless elements seems to be not powerful enough in its description and modelling capability. But if we assume that the components of an ontology are themselves structured, then what are the structures of these components? Our answer to this question is natural and straightforward, i.e., they have the structure of an object. That means, we just take objects as basic components of ontologies.

Let us use a metaphor: if we compare our paradigm with the globe where we are living, then the objects and their inheritance hierarchies can be compared with the meridian lines and the ontologies can be compared with the parallel lines.

Idea 4 (Let Ontologies Form Their Own Inheritance Hierarchies). It is well known that objects may form inheritance hierarchies. The methodology based on object hierarchies is called object oriented and that just based on a set of objects (without hierarchies) is called object based. If we have a look at the prevalent ontology approaches from this point of view, then most of them (as far as we have seen in the literature) can only be called ontology-based ones, not ontology-oriented ones.

What we want to do is to propose an ontology-oriented methodology that is based on hierarchies of ontology. This decision of us has brought many technical difficulties to overcome. They include how to define the inheritance mechanism of ontologies? What are the relations between object inheritance and ontology inheritance? Both questions are tightly related to the structures of ontology. It is easy to see that the inheritance of objects influences that of ontologies and vice versa.

Idea 5 (Allow Ontologies to be Nested). As it was said above, we take objects as basic elements of an ontology. But we may pose the question: is it possible that some ontologies may contain other ontologies as their components? Let us consider the example of chemistry once again. The atoms serve as components of molecules. Why can't the molecules serve as components of other more complex molecules, the macromolecules (high polymers)?

3 A Mathematical Model for Ontology

Now that the basic thoughts about the construction of ontologies have been explained, we are prepared to introduce a more formalized notation and definition for ontologies [①].

Definition 3.1 (Acyclic Cybergraph). *An acyclic cybergraph can be represented as a quin-tuple* (D, P, U, V, H)*, where the five sets* D*,* P*,* U*,* V*,* H *have no non-empty intersection with each other, in*

[①] In the following of this paper, those concepts concerning the object-oriented method are keeping their ordinary meanings.

which D is called the set of descriptors, P is called the set of processors, U and V are two finite sets of vertexes, called simple vertexes or complex vertexes, respectively, and H is a finite set of hyperplanes, where each hyperplane is determined by a subset of $U \cup V$, headed by a name denoting this hyperplane. Different hyperplanes have different names. A complex vertex is itself an acyclic cybergraph. Each element of $D \cup P \cup U \cup V \cup H$ is said to be contained in this acyclic cybergraph.

Definition 3.2 (Nested Cybergraph). *An acyclic cybergraph with a non-empty set V is called a* **nested cybergraph**. *A nested cybergraph (D, P, U, V, H) has the nesting level 1, if each cybergraph represented by any vertex of V is not nested. It has the nesting level $n+1$, if the maximal nesting level of the cybergraphs represented by the vertexes of V is n. An acyclic cybergraph, which is not nested, has the nesting level 0. For all $n \geqslant 0$, a cybergraph with nesting level n is called finitely nested.*

Two vertexes x and y of $U \cup V$ are called directly connected, if there is a hyperplane h of H, such that both x and y belong to h. Each element of $U \cup V$ is directly connected to the acyclic cybergraph it belongs to. The vertexes x and y are called connected if they are directly connected, or if there is another vertex z, such that x and z are directly connected and y and z are connected. A cybergraph is called connected if each pair of its vertexes is connected.

Definition 3.3 (Ontology Skeleton). *An* **ontology skeleton** *is a connected, finitely nested cybergraph, where the cardinality of $U \cup V$ is at least 2. The descriptors are called attributes. The processors are called methods. The simple vertices are called object skeletons* [②]*. The complex vertices are ontology skeletons themselves. And the hyperplanes are relations among the simple and complex vertices.*

Note that concrete ontologies may have many details and are thus not suitable for a formal description. Therefore, we define only an ontology skeleton formally. This definition should be enough for understanding the mechanisms of concrete ontologies.

Definition 3.4 (Homomorphic Mapping). *Let O_1, O_2 be two ontology skeletons, $O_1 = (D_1, P_1, U_1, V_1, H_1)$ and $O_2 = (D_2, P_2, U_2, V_2, H_2)$. We say that there is a* **homomorphic mapping** *M from O_1 to O_2, if*

(1) M is composed of five mappings from each component of O_1 to that in O_2:

$$M(O_1) = \{M(D_1), M(P_1), M(U_1), M(V_1), M(H_1)\}$$

(2) $M(D_1)$ is an identical mapping. That means, D_1 is a subset of D_2.

(3) $M(P_1)$ is an identical mapping. That means, P_1 is a subset of P_2.

(4) $M(U_1) = \{M(e_1) | e_1 \in U_1\}$, where for each $e_1 \in U_1$, there is one and only one $e_2 \in U_2$, such that [③]

(4.1) $M(e_1) = e_2 \geqslant e_1$; and

(4.2) $M(e_1) \neq M(e_2)$ for $e_1 \neq e_2$.

(5) $M(V_1) = \{M(e_1) | e_1 \in V_1\}$, where for each $e_1 \in V_1$, there is one and only one $e_2 \in V_2$, such that

(5.1) $M : e_1 \to e_2$ is a homomorphic mapping; and

(5.2) $M(e_1) \neq M(e_2)$ for $e_1 \neq e_2$.

(6) $M(H_1) = \{M(h_1) | h_1 \in H_1\}$, where for each $h_1 \in H_1$, there is one and only one $h_2 \in H_2$, such that

(6.1) $name(h_2) = name(h_1)$; and

(6.2) $cardinality(h_1) = cardinality(h_2)$; and

(6.3) $M(h_1) = \{M(hh_1) | hh_1 \in h_1\}$, where for each $hh_1 \in h_1$, there is one and only one $hh_2 \in h_2$, such that if hh_1 is an object class, then $M(hh_1) = hh_2 \geqslant hh_1$ and if hh_1 is an ontology skeleton then $M(hh_1)$ is a homomorphic mapping and $M(hh_1) \neq M(hh_2)$ for $hh_1 \neq hh_2$.

Property 3.5. *Let O_1, O_2, O_3 be three ontology skeletons. If there is a homomorphic mapping M_1 from O_1 to O_2 and a homomorphic mapping M_2 from O_2 to O_3, then the composition $M_2(M_1)$ is a homomorphic mapping from O_1 to O_3.*

[②] An object skeleton is indeed an object class.

[③] Let b_1, b_2 be two objects. We use the notation $b_1 < b_2$ (or, what is the same: $b_2 > b_1$) to denote the fact that b_1 is an ancestor of b_2. We also use the notation $b_1 \leqslant b_2$ (or, what is the same: $b_2 \geqslant b_1$) to denote ($b_1 = b_2$ or $b_1 < b_2$).

Definition 3.6 (Ancestor Relation of Ontology Skeleton). *Let O_1 and O_2 be two ontology skeletons. We say that O_1 is an* **ancestor** *of O_2, (or, what is the same, O_2 is an* **offspring** *of O_1) if there is a non-identical homomorphic mapping④ M from O_1 to O_2. O_1 is called a* **father** *of O_2, if O_1 is an ancestor of O_2, but there is no ancestor of O_2, which is an offspring of O_1.*

Definition 3.7. *Let $O_1 = (D_1, P_1, U_1, V_1, H_1)$ be a father of $O_2 = (D_2, P_2, U_2, V_2, H_2)$. Then O_2 has a short notation $O_2' = (D_2', P_2', U_2', V_2', H_2')$, in which*

(1) $D_2' = D_2 - D_1$, $P_2' = P_2 - P_1$;

(2) $U_2' = \{x' | \exists x \in U_2 - U_1, x'$ *is a short notation of* $x\}$;

(3) $V_2' = \{x' | \exists x \in V_2 - V_1, x'$ *is a short notation of* $x\}$;

(4) $H_2' = \{x' | \exists x \in H_2 - H_1, x'$ *is produced from x by replacing any ontology skeleton z contained in x with a short notation of z*$\}$.;

Definition 3.8 (Inheritance). *Let $O_1 = (D_1, P_1, U_1, V_1, H_1)$ be a father of $O_2 = (D_2, P_2, U_2, V_2, H_2)$. And let M be the homomorphic mapping from O_1 to O_2. If O_2', which is denoted by (D_2', P_2', U_2', V_2', H_2'), is the short notation of O_2, then the restoration of an ontology O_2 from its short notation O_2' is called* **inheritance**. *In this procedure, O_2' inherits from O_1:*

(1) *Anything of D_1, P_1, $U_1 \cap U_2$, $V_1 \cap V_2$ and $H_1 \cap H_2$;*

(2) *Anything of $z/M(z)'$ for $z \in U_1$, where y/x means those attributes and methods which belong to y, but not to x, $M(z)'$ means the short notation of $M(z)$;*

(3) *Anything of $z//M(z)'$ for $z \in V_1$, where $y//x$ means those things x inherits from y;*

(4) *Anything of $z//M(z)'$ for $z \in H_1$.*

A hyperplane inherits everything which its ontology skeletons and objects inherit.

Definition 3.9. *An ontology skeleton together with all its son ontologies is called an* **ontology skeleton class**. *This ontology skeleton is then called the root of this ontology skeleton class. All ontology skeletons of this class, which have no sons, are called leaves of this class.*

4 Architecture of Knowledge Model Base (KMB)

4.1 Design Principle of KMB

Definition 4.1 (Ontology). If $O = (D, P, U, V, H)$ is an ontology skeleton. Replace the elements in D and P by real attributes and methods. Replace the object skeletons in U and H by corresponding objects. Replace the ontology skeletons in V and H by corresponding ontologies. The result is called an **ontology**.

The KMB is composed of ontologies organized according to different business aspects and different application domains. During the design of KMB architecture, we have paid attention particularly to the following points:

• For the same application domain and the same kind of business, different managers will have different ideas for their information systems. We know that supermarket managers may have different opinions with respect to how detailed their information management should be. For example, their daily accounting should be made on a single piece ware basis or only made on the department or shop (for chained supermarkets) accounting basis? Which data should be included in their data tables? Do they need a data warehouse? If yes, how does it look like? Also the decision support models they use may be different. After all, the use of a certain information system means that in some sense people are following certain principles and strategies of management.

④Let $O_1 = (D_1, P_1, U_1, V_1, H_1)$, $O_2 = (D_2, P_2, U_2, V_2, H_2)$ be two ontology skeletons. A non-identical homomorphic mapping M from O_1 to O_2 is a homomorphic mapping from O_1 to O_2 satisfying the following extra constraints: (1) D_1 is a proper subset of D_2; or (2) P_1 is a proper subset of P_2; or (3) Let $e_1 \in U_1$, $e_2 \in U_2$. If $M(e_1) = e_2$ then $e_2 > e_1$; or (4) Let $e_1 \in V_1$, $e_2 \in V_2$. Then $M : e_1 \to e_2$ is a non-identical homomorphic mapping; or (5) Let $h_1 \in H_1$ and $h_2 \in H_2$. $M(h_1) = \{M(hh_1) | hh_1 \in h_1\}$, there is one and only one $hh_2 \in h_2$, such that if hh_1 is an object class, then $M(hh_1) = hh_2 > hh_1$.

• Some application domains may form hierarchies. For example, a domain model about the area of commerce business may have sub-models such as single shop models, supermarket models, chained shop models, etc.

• Different application domains may have overlapping domain knowledge. For example, every enterprise or government institution has a personal department. In most situations, a financial department is also indispensable.

• Definition of application domains is not unique. The concept *application domain* is quite vague. There is no unique standard about the division of application domains. There are even fewer standards about the business functions contained in some particular application domains. They are different with respect to countries and variable with respect to time.

• There are many factors which may influence the effectiveness of a domain model. Apart from the managers, the rapidly changing market situation affects also their use. And the government policies play also an important role, which determine the taxes, incomes, inputs, outputs of a country or of an area and thus directing the consumption of people. It would be not wise to include all of them in a single model.

4.2 Family of Knowledge Models

Definition 4.2 (Generalized Acyclic Cybergraph). *A* **generalized acyclic cybergraph** *is a six-tuple* (D, P, U, V, H, I)*, where* D*,* P *and* U *are the same as in Definition* 3.1*.* H *is a set of hyperplanes, where each node of each hyperplane is a simple node or a compound node. A simple node is an element of* $U \cup V$*. A compound node may be an and-node and*(S) *or an or-node or*(T)*, where* S *and* T *are subsets of* $U \cup V$*. An element of* V *is itself a generalized acyclic cybergraph.*

I *is a mapping. It maps each* $m \in H$ *to a non-negative integer number, called irrelevance degree of* m *with respect to this generalized acyclic cybergraph. Further, it maps each node* n *of each* $m \in H$ *to a non-negative integer number, called irrelevance degree of* n *with respect to* m*.*

As for the situation of compound nodes, each vertex v *in an and-node has the* **irrelevance degree** 0 *with respect to the node it belongs to. Each vertex* v *in an or-node has the* **quasi irrelevance degree** 0 *with respect to the node it belongs to. That means, none of the vertices contained in an or-node is necessary to that node (and to the hyperplane it belongs to), but the set of vertices in an or-node as a whole is necessary for that node.*

A generalized acyclic cybergraph with a non-empty set V is called a nested cybergraph. The concept of nesting in Definition 3.2 applies also for generalized acyclic graphs.

Two vertices x and y of $U \cup V$ are called directly connected, if there is a hyperplane $h \in H$, such that x and y are nodes of h or belong to the nodes of h. The vertices x and y are called connected if they are directly connected, or if there is another vertex z, such that x and z are directly connected and y and z are connected. A generalized cybergraph is called connected if each pair of its vertices is connected.

Definition 4.3. *Denote the* **irrelevance degree** *of* x *with respect to* y *as* $ID(x, y)$*, the quasi irrelevance degree of* x *with respect to* y *as* $QID(x, y)$ *and the upper bound as UB. Then we have*:

(1) $ID(x, y) = 0$ *if* y *is an ancestor of* x;

(2) $UB(ID(x, z)) = \max(ID(x, y), ID(y, z))$;

(3) *Point* 2 *is still valid if we replace* $ID(x, y)$ *with* $UB(ID(x, y))$ *and/or* $ID(y, z)$ *with* $UB(ID(y, z))$;

(4) *For any* x *contained in* y*,*

$$ID(x, y) = \min\{u | u \text{ is a } UB(ID(x, y))\}$$

(5) $UB(QID(x, z)) = \max(QID(x, y), QID(y, z))$;

(6) *Point* 5 *is still valid if we replace* $QID(x, y)$ *with* $UB(QID(x, y))$ *and/or* $QID(y, z)$ *with* $UB(QID(y, z))$;

(7) *Point* 5 *is still valid if we replace* $QID(x,y)$ *with* $ID(x,y)$ *or* $UB(ID(x,y))$*, or replace* $QID(y,z)$ *with* $ID(y,z)$ *or* $UB(ID(y,z))$*. But not both*;

(8) $QID(x,y) = \min\{u|\ u \text{ is a } UB(QID(x,y))\}$;

(9) *In all the cases mentioned above, we say that* x *has a finite irrelevance degree* (*or finite quasi irrelevance degree*) *with respect to* y*. In any other case, the irrelevance degree of* x *with respect to* y *is '*∞*'.*

Obviously, the larger this irrelevance degree is, the more the irrelevance is. If $ID(x,y) = n$, $n = 0$ means x is necessary for y. In the following we will see that this definition is meaningful.

Definition 4.4 (Weighted Ontology Skeleton). *A* **weighted ontology skeleton** *is a connected, finitely nested generalized acyclic cybergraph, where the cardinality of* $U \cup V$ *is at least* 2*. The descriptors are called attributes. The processors are called methods. The simple vertices are object skeletons. The complex vertices are weighted ontology skeletons themselves. And the hyperplanes are relations among the simple and complex vertices.*

Definition 4.5 (Sub-Homomorphic Mapping of Weighted Ontology Skeleton). *Let* $O_1 = (D_1, P_1, U_1, V_1, H_1, I_1)$*,* $O_2 = (D_2, P_2, U_2, V_2, H_2, I_2)$ *be two weighted ontology skeletons. We say that there is a* **sub-homomorphic mapping** M *from* O_1 *to* O_2*, if*

(1) M *is composed of six mappings from each component of* O_1 *to that in* O_2 :

$$M(O_1) = \{M(D_1), M(P_1), M(U_1), M(V_1), M(H_1), M(I_1)\}$$

(2) *The sub-mapping* $\{M(D_1), M(P_1), M(U_1), M(V_1),\ M(H_1)\}$ *is a homomorphic mapping from* $(D_1, P_1, U_1, V_1, H_1)$ *to* $(D_2, P_2, U_2, V_2, H_2)$ *defined in Definition* 3.4*, with the additional condition*:

(2.1) *each and-node and*(S) *is mapped to an and-node and*(T)*, where* S *is a subset of* T;

(2.2) *each or-node or*(S) *is mapped to an or-node or*(T)*, where* T *is a non-empty subset of* S*.*

(3) $M(I_1)$ *is an identical mapping such that* $I_1(a,x) = I_2(M(a), M(x))$*, where* $I_1(a,x)$ *is the irrelevance degree of* a *with respect to* x*.*

Similarly, we can define weighted ontologies and use the concept of sub-homomorphic mapping to define their inheritance and class hierarchies.

In the remains of this paper, we keep the name *ontology* instead of *weighted ontology*. But we should remember that the knowledge stored in our KMB is all in form of the weighted ontologies. Also, we use the name *entity* to stand for *ontology* or *object*.

Definition 4.6 (Connection Graph). *Let* g *be an ontology.* g *with all other entities which have a finite irrelevance degree* (*or quasi irrelevance degree*) *with respect to* g *forms a set, called the* **connection graph** *with* g *as its root.*

Definition 4.7. *Let* g *and* h *be entities. If the* (*quasi*) *irrelevance degree of* h *with respect to* g *is a finite number* n*,* h *is also called* (**quasi**) n**-connected** *to* g*. Additionally, each ancestor entity is* 0*-connected to all its offspring entities. If* a *is* n*-connected to* b*,* b *has an irrelevance degree* m *with respect to* c*, then* a *is* $\max(n,m)$*-connected to* c*.*

Definition 4.8 (Relevance Graph). *Given a set* S *of entities. For each* n *and each entity* $e \in S$*, all entities of* S*, which are at most* n*-connected to* e *form an* n*-connected* **relevance graph** *with* e *as its root.*

Note that the connection property is not symmetric. If h is n-connected to g, then g is not necessarily n-connected to h.

Definition 4.9 (Maximal Connection). *Let* K *be a KMB. If there is an integer* $m \geqslant 0$*, such that for any pair of entities* g *and* h *of* K*, where* h *is* n*-connected to* g*,* n *is not larger than* m*. And there is at least a pair* (g,h) *of entities, where* h *is* m*-connected to* g*. Then we call* m *the maximal connection of* K*.*

Note that a maximal connection exists for each KMB since KMB is always finite.

Definition 4.10 (Knowledge Model). *A* **knowledge model** *KM is a set of entities which exist in KMB in form of an* n*-connected relevance graph for a fixed* n *and with a fixed root, satisfying the following conditions*:

(1) *there is no $x \in KM$, such that x is contained in x;*
(2) *there is no $x \in KM$, such that x is an ancestor of x;*
(3) *there are no $x, y \in KM$, such that x is contained in y and there is an ancestor-offspring relation between x and y or between y and x;*
(4) *assume $H, G \in KM$ are ontologies. If entity a is contained in H, entity b is contained in G and H is an ancestor of G, then b is not an ancestor of a.*

The 'contain' relation and the 'ancestor-offspring' relation mentioned above are all meant in transitive sense. We say that KM has the extension degree n.

A KMB K is complete if with any entity $E \in K$, all of the entities mentioned in E exist in K. A KMB is perfect if each entity of K is contained in some knowledge model. Henceforth, we only consider complete and perfect KMB.

Let the root ontology R be fixed and let the extension degree n vary from 0 to m (maximal connection of the knowledge base), we get a series of knowledge models all of which have the same root R. We say also that R represents this series of models.

Example 1. We use $A, B, C, \ldots$ to denote ontologies, $P, Q, R, \ldots$ to denote hyperplanes, $a, b, c, \ldots$ to denote objects, a triple (α, n, β) to denote that the irrelevance degree of α with respect to β is n, a triple $[\alpha, n, \beta]$ to denote that α is n-connected to β (but α does not have a finite irrelevance degree with respect to β) and $\alpha(n)$ (in the body of an ontology definition) to declare that the irrelevance degree of α to its ontology is n. If

A contains $P(0)$	C contains $D(0), C(2)$
P contains $B(0), C(1), D(2), a(0)$	c is father of a
B contains $Q(0), R(2), b(1)$	a is father of b and d

Then we have

$((P, B, a, b, d, Q, D), 0, A)$	$((B, Q, D, a, b, d), 0, p)$
$(D, 0, Q)$	$((C, b), 1, (A, P))$
$(b, 1, B)$	$(a, 0, c)$
$(R, 2, (A, B, P))$	$(C, 2, (A, B, P, Q))$
$((b, d), 0, (a, c))$	$((Q, D), 0, B)$
$[c, 0, (a, b, d, P, A)]$	$[a, 0, (b, d)]$
$[c, 1, (A, B, P)]$	

5 Build Virtual Domain Models: A Genetic Approach

There are a big variety of business types in the real world. We are even justified to say that there are no two enterprises (companies) in the world, which are identical in all aspects of information processing. It would be nice to construct a model for each business type and to store all these models in KMB. But this is impossible. With the knowledge model base defined above, we can only keep the most basic models in KMB and try to build new ones whenever the user has the need. These new models which are built instantly are called virtual domain models since they do not belong to the set of most conventional models in the relevant field.

Definition 5.1 (Information System Ontology Skeleton). Information system ontology skeleton *is an ontology skeleton which is structured as* $(\{EntInf\}, \{\}, \{\}, \{Org, Dat\}, \{Use(Org, Dat)\})$.

This is an abstract ontology skeleton (class or type) for information systems. In which, *EntInf* is the type descriptor, *Org* and *Dat* are two other ontology types it contains. '*Use*(*Org*, *Dat*)' means that there is a relationship '*Use*' between ontology *Org* and *Dat*. This ontology skeleton means that any information system contains two parts: organization and data, and this system is for capturing how organization use data.

Definition 5.2 (Domain Model). *A* **domain model** *is a knowledge model (for a fixed extension degree n) with an information system ontology as its root.*

Definition 5.3 (Virtual Domain Model). *A* **virtual domain model** *VDM is a domain model (for a fixed extension degree n) and with at least one of the following characteristics*:

(1) *All of its entities exist in KMB, but they are not forming an n-connected relevance graph*;

(2) *Some of its entities do not exist in KMB, but each of them is a variation of some entity in KMB, where entity A is called a variation of entity B, if A and B have the same name and the same father, but with different contents*;

(3) *Some of its entities come from outside by user input.*

We also say that this VDM has the extension degree n.

In the rest of this section, we give the skeleton of our approach for constructing virtual domain models. Detailed description of these methods can be found in [12]. In this approach, we consider the KMB as a world of domain models, which, like the living things, form a population. This population will experience a procedure of evolution, multiplying, propagation and development during the process of software reuse. This population of domain models grows steadily together with the change of the reuse environment. It comes also closer and closer to the real world of application domains. For the reason explained above we make use of terminology from genetics for the classification of these techniques. They are selection, clone, mutation, crossover, synthesis and transgenic production.

Definition 5.4. *The basic operations for constructing a virtual domain model include*:

(1) **Selection** *means to chose one or some existing domain models or entities, which are expected to serve as the best raw material for further elaboration.*

(2) **Clone** *means to copy an existing entity or an intermediate product to be further elaborated into the final product.*

(3) **Mutation** *means to change the content of a selected entity or an intermediate product.*

(4) **Crossover** *means to combine (whole or part of) two or more domain models to get a new one.*

(5) **Synthesis** *means to integrate several smaller entities to a larger one.*

(6) **Transgenic** *means to add (or delete) some entities to (or from) a domain model to get a new one.*

In the following, we give some case studies to show that these operations are meaningful.

Case Study 1 (Clone). Let R be an ontology of information system ontology skeleton and n an integer number. Required is to construct a domain model with R as the root and n as the extension number.

Example 2. The user wants a domain model for three-star hotels, i.e., $R =$'hotel information system' and $n = 3$.

Sometimes, we want to build a domain model by combining two existing domain models. As we will see below, this is a complicated problem and the required operation will raise a lot of technical problems. At this place, we are satisfied with the solution of only a humble sub-problem of it.

Case Study 2 (Crossover). Given are two names R and T representing two series of domain models. Required is to construct a minimal domain model W which contains one model from each of the two series.

Example 3. The user wants to combine the domain models of hotel and shopping center.

Sometimes we do not want to start from a given domain model, but want to construct a new domain model from a given set of knowledge units.

Case Study 3 (Synthesis). Given is a set S of entities. Required is to construct a domain model DM which contains all entities of S.

Case Study 4 (Transgenic-1). Given is an ontology R of information system ontology skeleton and a set S of entities. Required is to construct a domain model with R as root and containing S.

Example 4. A bank has the business of accepting deposits and providing credits. A post office has the business of mail and remittance service. Now the Chinese post offices have opened a new service of accepting deposits, which is originally only a part of the bank business.

Case Study 5 (Transgenic-2). Given are a domain ontology knowledge base K with the maximal connection m, an ontology R of information system ontology skeleton, a non-negative number $n \leqslant m$

and a set S of entities. Required is to construct a domain model with R and extension number n, which includes S.

Example 5. The user wants a domain model of three-star hotels with shopping centers. (Assume that the user does not know whether a three-star hotel normally has a shopping center or not).

Sometimes we do not want to add some knowledge units to a domain model, but on the contrary, we want to delete some knowledge units from it.

Case Study 6 (Transgenic-3). Given is the name R of an ontology of information system ontology skeleton and two sets S and T of entities. Required is to construct a domain model with R as root, which contains S, but does not contain T.

Example 6. Some 'sleep-inn' type hotels do not provide restaurant service.

Case Study 7 (Transgenic-4). Given is the name R of an ontology of information system ontology skeleton and an entity E, which is not n-connected to any domain model with R as root for any extension degree n. Required is to construct a domain model with R as root and containing the entity E.

Note that this entity E does not necessarily exist in the knowledge base before. It can be added to the knowledge base by the user on the spot.

Example 7. The manager of a supermarket wants to have a new kind of daily business report which may mean a new data view object and a new data processing object.

Case Study 8 (Transgenic-5). Given are the name R of an ontology of information system ontology skeleton and a finite set of n-connected relevance graphs, each with an entity as root. All roots of these relevance graphs are not n-connected to any domain model with R as root for any extension degree n. Required is to construct a domain model with R as root and containing all these relevance graphs.

Example 8. A computer company is going to expand its business area to include some new services such as local area network integration, supermarket MIS development and electronic business consultation.

Case Study 9 (Transgenic-6). Given is a domain model DM with R as root and an extension degree n. Let E be an entity, which is j-connected to R. We have $j \leqslant n \leqslant m$, where m is the maximal connection of the KMB. Required is to construct a sub-model of DM with R as root and not containing the entity E.

Example 9. A special tax sort of the Haidian District has been removed. Consequently, the MIS of the Haidian tax bureau does not need to print a table for this tax each month.

Case Study 10 (Transgenic-7). Given a domain model DM with R as root. Given are also a finite set S of entities. Each of these entities is j-connected to DM for some j, $j \leqslant n \leqslant m$, where m is the maximal connection of the KMB. Required is to construct a minimal sub-model of DM with R as root and not containing any of these entities.

Example 10. A supermarket will close its video product department, computer device department and camera and fine instrument department. This would not mean, however, that anything involved in the information processing of these three departments should be deleted. For example, some invoice formats of the three removed departments are the same as those used in other departments. We delete only those elements that are only needed by these three departments, but not needed by other ones.

6 Conclusions

Nowadays, there exist many ontologies[26,27]. In summary, we can get the following main types of ontologies.

• Knowledge representation ontologies capture the representation primitive used to formalize knowledge in knowledge representation paradigms. The representative example is the Frame-Ontology[28] which has been implemented in KIF 3.0[29].

• General ontologies include vocabularies related to things, events, time, space, causality, behavior, function, etc. The CYC ontology[30] is a common sense ontology.

• Domain ontologies are reusable in a given domain. They provide vocabularies about the concepts within a domain and their relationships. The EngMath[31], PhysSys[32] and TOVE are well-known domain ontologies.

• The linguistic ontologies contains linguistic concepts and the relationships between these concepts. The Generalized Upper Model[33] and WordNet[34] are most illustrative linguistic ontologies.

• Task ontologies[11] provide a systematic vocabulary of the terms used to solve problems.

As our formal ontology is designed for representing the domain knowledge resulted by domain analysis, we mainly focus on the structure of the domain. We provide mechanism for representing domain concepts and various relationships between concepts. This formalization reveals that the conceptualization of the domain models could be represented as a multiple layer conceptual connection graphs. This kind of formal structure is the prerequisite for automatic knowledge sharing and reusing, i.e., knowledge sharing between computers or computer systems.

In terms of the above categories, what we have done in this paper could be categorized as a knowledge representation ontology. Different from the most representative knowledge representation ontology, i.e., Frame-Ontology which is implemented in KIF 3.0 based on first-order logic, our ontology combines the algebra approach and the concept graph approach, that makes our ontology representation more intuitive and more structured. On the one hand, we extend the concept graph by allowing the concept nodes having their own structure. Each concept node is an object and has the structure of object. On the other hand, we use algebra mapping, i.e., the homomorphic mapping, to capture the inheritance hierarchy of ontologies in different levels of abstraction. So, we have two kinds of inheritance, i.e., object inheritance and ontology inheritance. The former is inheritance for concepts and the latter for structure. This feature makes our ontology representation more flexible. Of course, the inheritance mechanism is also very complicated. But that is beyond the scope of this paper and will be one of our further research topics.

And finally, based on this formalization, some operations for sharing and reusing domain knowledge have also been discussed.

References

[1] Maymir-Ducharme F A. Varying domain engineering approaches — Business case perspectives. In *Proceedings of 5th Annual Workshop on Software Reuse Education and Training,* Morgantown, WV, 1996.

[2] Neighbors J. Software construction using components [Dissertation]. Information and Computer Science Department, University of California, 1981.

[3] Domain Analysis. Technical Report, 1997. available from http://WWW.sern.enel.ucalgary.ca/charmi/Courses/97-98/ENEL619.02/FuSeminar.html

[4] Prieto-Diaz R, Arango G. Domain Analysis and Software Systems Modeling. IEEE Computer Society Press, Los Alamitos, California, 1991.

[5] Simos M A. Organization domain modeling (ODM) guidebook. Technical Report STARS VC-A023/011, Unisys Corporation, 1995.

[6] Kyo C Kang, Sholom G Cohen, James A Hess, William E Novak, A Spencer Peterson. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Pittsburgh, PA: Software Engineering Institute, Carnegie Mellon University, 1990.

[7] McCain R. Reusable software component construction — A product oriented paradigm. In *Proceedings of AIAA/ACM/ NASA/IEEE Computers in Aerospace Conference,* 1985, pp.125–135.

[8] Yi X. Study on Software Development of Special Domains based on Architectures [Dissertation]. Peking University, 1996.

[9] Yang F. Overview of the jade bird project. Technical Report, Department of Computer Science, Peking University, 1997.

[10] Capilla R. Application of Domain Analysis to Knowledge Reuse. In *Proceedings of the 8th Annual Workshop on Institutionalizing Software Reuse* (*WISR'8*), Ohio State, USA, 1997. http://citeseer.nj.nec.com/capilla97 application.html

[11] Chandrasekaran B. Generic tasks in knowledge-based reasoning: High level building blocks for expert system design. *IEEE Expert,* 1986, 1(3): 23–30.

[12] Lu R, Jin Z. Domain Modelling-Based Software Engineering. Volume 8 of The Kluwer International Series on Asian Studies in Computer and Information Science, Kluwer Academic Publishers, 2000.
[13] Lu R, Jin Z. Hierarchical software reuse. *Chinese Journal of Advanced Software Research,* 1999, 6(1): 1–11.
[14] Lu R, Jin Z, Wan R. PROMIS: A knowledge-based tool for automatically prototyping management information systems. In *Proceedings of AVIGNON'94,* Paris, France, 1994, 325–330.
[15] Lu R, Jin Z, Wan R. Requirement specification in pseudo-natural language in PROMIS. In *Proceedings of 19th COMPSAC,* USA, 1995, pp.96–101.
[16] Lu R, Jin Z, Wan R, Xie Y. Domain knowledge based requirements information acquisition. *Journal of Software,* 1996, 7(3): 137–144.
[17] Lu R, Jin Z, Liu L, Jiang A, Lai H. PROMIS 2.0: An intelligent CASE tool for MIS in the client/server application. In *Proceedings of the 2nd International Symposium on Future Computer Systems,* Xiamen, P.R. China, 1997, pp.399–405.
[18] Lu R, Jin Z, Liu L, Fan G, Chen G, Xun X, Wang S. OSNET: A Language for Domain Modeling. In *Tools Asia'98,* Beijing, P.R. China, 1998, pp.198–203.
[19] Lu R, Jin Z, Liu L, Jiang A, Lai H. NEWCOM: An architecture description language in client/server style. *Chinese Journal of Computers,* 1998, 21(12): 1103–1111.
[20] Uschold M, Gruninger M. The enterprise ontology. Technical Report AIAI-TR-195, Artificial Intelligence Application Institute, University of Edinburgh, 1997.
[21] TOVE Project. TOVE Manual. Technical Report, Enterprise Integration Laboratory, University of Toronto, available from http://www.ie.utoronto.ca/EIL/tove, 1995.
[22] Chen P. The entity-relationship model — Towards a unified view of data. *ACM Transactions on Database Systems,* 1976, 1(1): 9–36.
[23] Sowa J F. Conceptual Structure: Information Processing in Mind and Machine. Addison-Wesley Publisher, Reading, Mass, USA, 1984.
[24] IEEE Standard for Developing Software Life Cycle Processes. IEEE Computer Society, New York, 1996.
[25] Fernandez M. Overview of methodologies for building ontologies. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods,* 1999, pp.1–13.
[26] A Gomez-Perez, V Richard Benjamins. Overview of knowledge sharing and reuse components: Ontologies and problem-solving methods. In *Proceedings of the IJCAI-99 Workshop on Ontologies and Problem-Solving Methods* (*KRR5*), Stockholm, Sweden, 1999.
[27] Natalya Fridman Noy, Carole D Hafner. The state of the art in ontology design. *AI Magazine,* Fall, 1997, pp.53–74.
[28] Gruber T R. Towards principles for the design of ontologies used for knowledge sharing. Technical Report KSL-93-04, Knowledge Systems Laboratory, Stanford University, 1993.
[29] Genesereth M R, Fikes R E. Knowledge interchange format. Technical Report KSL-92-86, Knowledge Systems Laboratory, Stanford University, 1992.
[30] Lenat D B. CYC: A large scale investment in knowledge infrastructure. *Communication of ACM,* 1995, (11): 33–38.
[31] Gruber T R, Olsen R. An ontology for engineering mathematics. Technical Report KSL-94-04, Knowledge Systems Laboratory, Stanford University, 1994.
[32] Borst W N. Construction of Engineering Ontology [Dissertation]. University of Twente, Enschede, 1997.
[33] Bateman J A, Magnini B, Fabris G. The generalized upper model knowledge base: Organization and use. Towards Very Large Knowledge Bases: Knowledge Building and Knowledge Sharing, Mars N J I (ed.), IOS Press, 1995, pp.60–72.
[34] Miller G A. WordNet: A lexical database for English. *Communications of ACM,* 1995, 39–41.

LU Ruqian, male, was born in Shanghai, China, in 1935. He received the degree of Diplom Mathematiker from the Friedlich Schiller University of Jena in 1959. He joined the Institute of Mathematics in 1959 and is now a professor of this institute. He is also a professor of Fudan University and a guest professor of Institute of Computing Technology. He is a academician of the Chinese Academy of Sciences. His research interests include artificial intelligence, knowledge engineering and knowledge science, and theoretical computer science.

JIN Zhi is a professor of computer science at the Academy of Mathematics and System Sciences. She received her Ph.D. degree in computer science from National University of Defense Technology in 1992. Her research interests include knowledge-based systems, ontology development, artificial intelligence, and requirements engineering. She has authored and co-authored one book and over 50 papers.