Beyond Knowledge Engineering

Ru-Qian Lu[1,2,3,4] (陆汝钤) and Zhi Jin[1,2] (金　芝)

[1]*Academy of Mathematics and System Science, Chinese Academy of Sciences, Beijing 100080, P.R. China*

[2]*Institute of Computing Technology, Chinese Academy of Sciences, Beijing 100080, P.R. China*

[3]*Shanghai Key Lab of Intelligent Information Processing, Fudan University, Shanghai 200433, P.R. China*

[4]*Beijing Key Lab of Multimedia and Intelligent Software, Beijing University of Technology, Beijing 100083, P.R. China*

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

Received March 29, 2006; revised July 17, 2006.

Abstract Knowledge engineering stems from E. A. Figenbaum's proposal in 1977, but it will enter a new decade with the new challenges. This paper first summarizes three knowledge engineering experiments we have undertaken to show possibility of separating knowledge development from intelligent software development. We call it the ICAX mode of intelligent application software generation. The key of this mode is to generate knowledge base, which is the source of intelligence of ICAX software, independently and parallel to intelligent software development. That gives birth to a new and more general concept "knowware". Knowware is a commercialized knowledge module with documentation and intellectual property, which is computer operable, but free of any built-in control mechanism, meeting some industrial standards and embeddable in software/hardware. The process of development, application and management of knowware is called knowware engineering. Two different knowware life cycle models are discussed: the furnace model and the crystallization model. Knowledge middleware is a class of software functioning in all aspects of knowware life cycle models. Finally, this paper also presents some examples of building knowware in the domain of information system engineering.

Keywords knowledge engineering, knowware, knowware engineering, knowledge middleware, domain knowware

1 Challenges to Traditional Knowledge Engineering

When E. A. Feigenbaum proposed the term knowledge engineering in 1977, nine years have elapsed since the concept software engineering was born on the NATO conference in 1968. However, at that time people's speculation about the scope and dimension of knowledge engineering was far less ambitious than those of software engineering. The typical scenario can be described as follows: a group of few knowledge programmers (also called knowledge engineers) finding and visiting some expert of a well limited domain, listening to the expert's introduction, writing down his/her experiences in some form of protocol, constructing a knowledge base counted from a dozen till thousands of rules, combining it with an inference engine. After that, a new expert system is produced.

Nearly thirty years have passed since the pioneering work of Feigenbaum and his colleagues. Knowledge engineering has made immense progress and produced considerable economic and social benefits. This is also true in China. First Chinese expert system, working in the field of traditional Chinese medicine (Dr. You-Bo Guan's experience in diagnosis of hepatitis) was developed in the late 1970s. In the last thirty years, we have seen countless projects, papers and books reporting advances of knowledge engineering in China.

Three years ago, a special issue of Journal of ACM published 15 papers written by Turing award and Nevalinna prize winners. In one of these papers Feigenbaum suggested three challenging problems[1]:

1) let computer pass the Feigenbaum test, a weakened form of Turing test, limiting it to some field of domain knowledge;
2) let computer read massive literature and transform it to a huge knowledge base, reducing the workload of knowledge engineering to 1/10;
3) let computer read massive web pages and transform them to a huge knowledge base, reducing the workload of knowledge engineering to 1/100.

Open and massive knowledge obviously implies new research direction. There is enough evidence showing that knowledge engineering has entered a new era which is marked by non canonical knowledge processing in an open and massive framework. With non canonical knowledge we mean knowledge that is intentionally intractable. Knowledge that is fuzzy, inexact, inconsistent, noisy, context dependent, etc. belongs to this category. As a matter of fact, all of them have been studied by computer scientists since long time ago. The only difference is that now we must solve real problems involving a massive amount of such knowledge in an open and steadily changing environment, while previously only theoretical research or limited size experiments in closed and static environment were enough. Examples of massive size knowledge engineering are Lenat's CYC[2] and Cao's NKI[3]. Open knowledge engineering is a theme

Regular Paper

Partly supported by the National Natural Science Foundation of China (Grant Nos. 69733020, 69983010, 60233010 and 60496324), the National Grand Fundamental Research 973 Program of China (Grant No. 2002CB312004), the Knowledge Innovation Program of the Chinese Academy of Sciences and MADIS of the Chinese Academy of Sciences.

that is still relatively young. A typical example is web knowledge processing. As examples, we can list semantic web[4], knowledge grid[5], P2P research[6], etc. All of them have made remarkable progress, which is nevertheless far from satisfying the requirements of the information society era.

On the other hand, our research in the past has shown that focusing on knowledge within a limited domain is a reasonable choice, like revealed by Feigenbaum's first challenging problem. Lots of domain knowledge bases, or more general domain ontologies have been built and some knowledge-based or ontology-based solutions have been proposed in many areas for automating the traditional solutions and making them more intelligent, but few of them could serve as a basis to pass the Feigenbaum's test. The difficulties include fully structuring and formalizing the domain knowledge and making them operatable in a concordant way with the meaning of the knowledge by computer programs.

This paper aims to reveal some hints on these two issues by presenting some experiments that we have done in the past decades. We also try to show that a new decade of the research on knowledge engineering is coming. To meet the challenge of this era, we propose the concepts knowware and knowware engineering. In the rest of this paper, Section 2 reports some experiments on automatic generation of knowledge-based systems and proposes the concept knowware. Section 3 illustrates the process model of knowware engineering. Section 4 presents the knowware on the organization information systems which we have built for supporting the automation of the whole process of the knowledge based information system development. Section 5 concludes the whole paper and makes some comments on this research direction.

2 From Knowledge Base to Knowware

2.1 Experiments in Massive Size and Open Source Knowledge

In the past 18 years, we have done a series of researches on automatic generation of knowledge based systems. The main difficulty in these efforts is how to construct a consistent and relatively complete knowledge base. As it was shown by practice, this job cannot be done easily by relying only on knowledge elicitation from some domain expert, or by knowledge acquisition from technical literature with pure natural language understanding techniques. We decided to combine both in the way of making the knowledge acquisition process a two staged one. For that purpose, we developed a technique of Pseudo-Natural Language Understanding, PNLU for short[7]. A language is called pseudo-natural if it is a subset of some natural language and has a well-defined grammar. Its terminal/non-terminal symbols are called keywords/parameters. The pattern of keyword sequence in a sentential form is called a keyword expression. A language parsing is called PNLU, if only the keywords, but not the other symbol strings between keywords, are parsed and understood. Given a technical literature, the semantics of each sentence, each segment and even the whole text is determined by the keywords and keyword expressions contained in it. For example, the sentential form

$$\langle A \rangle \text{ is classified in } \langle B \rangle \text{ and } \langle C \rangle$$

may recognize the sentence "blood cell is classified in red blood cell and white blood cell", where the parser does not analyze the meaning of the three "blood cell" parameters. This is enough for constructing a knowledge base. A knowledge compiler from PNL text (considered as a program) to some knowledge representation, such as frames, rules or productions was developed. In the first step, we adapt the given technical literature to PNL form manually, which requires only a little work. The PNL text will be then compiled into a knowledge base automatically. Combining this knowledge base with some existing inference engine, we get a knowledge based system ready to be run. In the second stage, we present this "rough" knowledge based system to some domain expert who checks it with a set of test cases. With repeated knowledge refinement and improvement the experts can turn the "rough" system into a refined one. We call this two staged process as a rational work division between human and computer. While the former reduces the difficulty of natural language understanding, the latter easies the formidable burden of extracting and summarizing knowledge from a huge amount of technical literature.

Having got the idea illustrated above, we have performed several experiments to check its feasibility in the long run of about 15 years. Our experiences coming from these experiments might be able to answer some of the challenges proposed by Feigenbaum. It is just based on these experiences that we have come to the concept of knowware and knowware reader, which we think may become a promising direction of computer science research and knowledge industry.

The first experiment was done on automatic construction of expert systems[8]. The PNL is called BKDL (Book Knowledge Description Language) with a set of about 250 keywords and basic expressions. They are divided into three levels: the system level, the domain level and the user level. Higher level keywords can be defined with lower level ones. BKDL mainly applies to classification type of knowledge. The BKDL texts are transformed to internal knowledge representation characterized by frames, including goal frames, hypothesis frames and attribute frames, together with a dictionary of terms and a user interface lexicon. With a built-in inference engine and some manual work in addition, the knowledge compiler was able to transform a small medical handbook to an expert system with reasoning and explanation facilities.

The second experiment was done on automatic construction of intelligent tutoring systems (ITS), which require more technical *subtleties* than constructing expert systems. First, the ITS knowledge base must be organized in the teaching-oriented manner. Second, this organization is user independent, but user tailored course material can always be generated based on user requirements. Third, during each session, student model can be automatically generated and dynamically adjusted. Fourth, the generation of exercises and tests should be automatic. We have developed an automatic ITS generation tool, called KONGZI[9], which can work in Chinese or English mode. The PNL for Chinese (or English) version is EBKDL, i.e., Educational BKDL, (or SELD, i.e., Scientific English Literature Description). The most important advantage of our approach is the automatic generation of ITS knowledge base, which to our knowledge does not exist in other prevalent ITS techniques. KONGZI has already been used for generating ITSs in medicine, biology and industry.

The third experiment was the PROMIS project[10] targeting at automatic construction of management information systems. The corresponding PNL is called BIDL (Business Information Description Language). This time the knowledge representation mechanism is a hierarchy of representation languages. The BIDL text will be first transformed into ONONET (ONtology and Object-oriented NETwork) form to get a conceptual model. This language combines the concept of objects and ontologies with semantic networks, while still keeping the flavor of the object-oriented paradigm. It serves now as the principal representation of domain knowledge in the knowledge base. Then the automatic planning of target system architecture takes place and produces an architecture design in NEWCOM (NEW COnceptual Model. To be exact, NEWCOM is a language for describing, modeling and implementing software architectures based on client/server paradigm. Generally speaking, a program in NEWCOM is a description of a local network in client/server style. It consists of a set of local connective networks and a set of client nodes and server nodes in each network.

2.2 From X Knowledge Base to Knowware

Summarizing the experiences of the above three projects shortly reported above, we can learn something about the rules of constructing knowledge based systems automatically. Usually we denote an application program in some domain X as a CAX (computer aided X) program. Further we call it an ICAX program if its function shows some intelligence. Actually, a knowledge based system is an intelligent system and the source of intelligence is knowledge. Thus we get the equation:

$$\text{ICAX} = \text{CAX} + X \text{ knowledge base}$$

which proposes a paradigm of intelligent application software development. This paradigm has the following consequences:

1) For any ICAX system, its X knowledge base can be developed separately and independently from the development of the CAX system itself.

2) Therefore, they can be produced by different people and groups, provided that the standards are observed.

3) In particular, each X knowledge base can play the role of an independent commodity, as it has been pointed out by E. A. Feigenbaum: "Knowledge itself will be a salable commodity like food and oil. Knowledge itself is to become a new wealth of the nations"[11]. A commodity is called independent if it needs not be sold as a part of another commodity containing it. As a consequence, the users may buy a CAX platform from company A and an X knowledge base from company B separately.

4) Since the knowledge-independent functions of a CAX system are relatively stable, once the basic architecture of this CAX system is given, the key of generating (renewing) its ICAX version automatically is the automatic generation of the corresponding (new) X knowledge base.

So it is quite worthwhile and interesting to study the essential properties of X knowledge bases and their generation procedures. We propose to study and develop knowware—the standardized and commercialized form of X knowledge base. More exactly, knowware is a commercialized knowledge module that is computer operable, but free of any built-in control mechanism (in particular, not bound to any software), meeting some industrial standards and embeddable in software and/or hardware[12]. Furthermore, a knowware includes a documentation and is connected to an intellectual property. Roughly speaking, in some (not all) aspects, the difference between the concepts of knowware and knowledge base is similar (or analog) to that between the concepts of software and program.

There are many examples of knowware, however currently mainly in their naive forms. Imagine an MP3 player, which is a combination of hardware and software. All the songs collected in it can be considered as knowware. The second example is tax calculating software. The tax regulations (considered as the declarative part of tax calculation) published by the government and operated by this tax calculating software form a knowware. The third example is economy situation evaluator whose job is to produce economy situation evaluation reports. As knowware we need a summarized account of market quotations and expert reviews and a knowledge base of economics theory. Let us compare these three cases of knowware. In the first example, there is no need to transform the songs to some particular form of knowware, since their representation is already standardized. In the second example, the tax regulation text has to be transformed to being in computer operable form. For this purpose, we need two standards: the standard of tax regulation texts and that of their knowware representation. We need also a compiler for transforming the first standard to the second one. As a matter of fact, we expect that the governmental tax departments will publish their tax regulations in two forms at the same time: a paper version for human reading and a knowware version for computer use. All that the system manager has to do is buying this new knowware and installing it on their tax software. In the third example we need two kinds of knowware. Besides, we are encountering a new difficulty: the knowledge is distributed widely and we even do not know where the knowledge sources are. To gather such knowledge, we need tools for knowledge acquisition. For collecting knowledge from the Internet, the web browser is a possible candidate. But we will see that we need far more delicate tools to do this job than the prevalent browsers.

Now we come to the architecture and representation of knowware itself. Roughly speaking, a knowware ready for use is a knowledge crystal meeting the needs mentioned above (see the explanation below). There are many forms of representation a knowledge base can take. Consequently, there are equally many forms of architecture and representation a knowware may take. For video products like MP3 players, one has to follow the standard of industry. There is not much space for innovation. It is simple and strict. For knowware like tax regulations in the second example, we noticed that it may be difficult to transform a natural language text directly into some machine operable knowledge representation. In this case, the PNL explained above may be a good candidate of intermediate representation. Each time when new tax regulations are given, it is easy to translate their NL text form into PNL text form with only minor efforts. Everybody, not just experts, can do it with a little training. The PNL interpreter then compiles it to a set of knowledge items in some machine operable representation. The question is what we should do with the old knowware representing tax regulations that are no more in force? Should the new knowware replace the old one? Or it just revises the old one? In many cases, a total replacement may present a good choice, since it is simple and raises no consistency problems. In case that the new knowledge is just a revision to the old one, the revision is not made on knowware itself, but on its semi-finished prototype: the knowledge crystal.

The knowware mentioned in the third example has a very wide area of application. Its knowledge is collected, fused and improved continuously over the time. Consider a vast source of knowledge like the World Wide Web. The knowledge mining process on it is just like knowledge crystallization from a knowledge solution. The crystallization core is a knowledge model in the relevant domain. This process is not just a monotonic piling up of knowledge items. Each time a new knowledge item is acquired, an evolution of the old crystal follows. Similar knowledge items may be merged. Complimentary items may be fused. Inconsistent items may be resolved. The whole crystal may be reorganized. But all of this is not a must. Only "knowledge" proved to be false must be deleted. This shows that a knowledge crystal is in a state of steady changing. We keep two different forms of knowledge organization, knowledge crystal and knowware, due to the following reasons:

1) Since knowledge crystal is always evolving, we do not require it to be consistent and complete. Every knowledge item in it is correct under some circumstance. That is enough.

2) Confronting knowledge items are not necessary useless. They may be valid under different circumstances. Therefore, it would be a lost to force a crystal to be consistent, since in order to do so one would have to delete one from two confronting items.

3) Usually, the same knowledge can be used in different domains, different aspects and different ways. Therefore the content and organization of knowledge in a crystal should be general purposed. On the other hand, a knowware may very well be some particular application oriented. Its content selection and knowledge organization may depend very much on this special purpose.

4) Each time when a knowware is to be produced, a consistent subset of knowledge items is selected from the crystal according to some knowware specification that is defined as a set of constraints and transformations on the crystal's knowledge items. The resulted knowledge items will be reorganized and transformed into the final knowware form. They are complete in the sense that the resulting knowware is enough for the intended use.

5) Knowware does not evolve. It can only be replaced with new knowware.

Table 1 lists the differences among knowware, knowledge crystal and knowledge base.

Table 1. Comparing Knowware with Knowledge Crystal and Knowledge Base

	Knowware	Knowledge crystal	Knowledge base
Equipped with a management system	No	No	Yes
Is a commodity	Yes	No	Not necessary
Has intellectual property	Yes	Not necessary	Not necessary
Includes a user documentation	Yes	No	Not necessary
Meets industrial standard	Yes	No	No
Mechanism of knowledge evolvable	Substitute its old knowledge content with a new knowledge crystal	Evolution based on life cycle model	Arbitrary, possibly with knowledge maintenance

3 From Knowledge Engineering to Knowware Engineering

3.1 Knowledge Middleware and Knowware Engineering

The development, application and management of knowware are a complicated process, which is never less complicate than the development, application and management of software. It involves knowledge acquisition, selection, fusion, maintenance and renewing. We need software tools, called knowledge middleware, for performing these jobs. For explaining the meaning of knowledge middleware let us first recall the definition of middleware in software engineering: "Middleware is the underlying software based on operating systems with basic communication protocols, whose function is to support the effective development, installation, running and management of application programs". Thus, traditional middleware helps application programs to work cooperatively in a networked environment. The operation of knowware needs a network in a broader sense than the communication network controlled by operating systems. This functional network connects not only knowware with knowware, but also knowware with software, knowware with knowledge source and knowware with human users. We call it the knowledge broker network, KBN for short. We can then modify the above definition of middleware to get a definition for knowledge middleware: knowledge middleware is the underlying set of software tools based on KBN and knowledge transformation and transmission protocols, whose function is to support the effective development, application and management of knowware.

Now let us consider the three knowware examples listed above once again. What kinds of knowledge middleware could be of use for these knowware? In the case of MP3 player like video products, the various programs running on P2P grids for searching and downloading music pieces are actually knowledge middleware. Furthermore, those programs for controlling and charging P2P music users belong also to the range of knowledge middleware. In the example of tax regulation knowware, the PNL interpreter compiling PNL texts to machine operable representation is a kind of knowware. In the example of economy situation evaluator, we need many kinds of knowledge middleware. For producing knowledge crystal we need, for example, intelligent browser, knowledge distiller, knowledge fusser and so on. For producing know-ware itself we need knowledge extractor, contradiction resolver, knowledge editor and so on. A knowledge distiller collects knowledge pieces from distributed, unorganized and even implicit knowledge sources, while a knowledge extractor acquires structured knowledge modules from a knowledge crystal. Besides, knowware users need a software called knowware reader, whose function includes statistics calculator, diagram drawer, particular knowledge extractor, etc. A more detailed illustration will be given in the next section.

Roughly classifying, we have the following kinds of knowledge middleware (KM for short): KU (Knowware-User) type KM: those helping the people to make use of knowware and helping the administrators to manage such use; CS (Crystal-Source) type KM: those functioning in the crystallization process of knowledge; CC (Crystal-Crystal) type KM: those functioning in the evolution process of knowledge crystal; CK (Crystal-Knowware) type KM: those transforming knowledge crystal to knowware; KK (Knowware-Knowware) type KM: those combining several knowware to a more powerful knowware.

Now we come to the concept of knowware engineering. We define knowware engineering as the systematic application of knowledge middleware with the goal of knowware generation. Knowware engineering has life cycles, just as software engineering does. Depending on how one obtains knowledge, to organize it in knowledge crystals, maintain it, make it evolving and transform it to knowware, one has different kinds of life cycles for knowware engineering.

Again consider the three examples discussed above. The first example of MP3 player makes use of mature technology (of coding a song) directly. There is no need to have a complicated life cycle. In the second example, we note that tax regulations are only a part of government policies. There are many others, for example salary regulations, unemployed regulations, insurance regulations, etc. Each of them becomes a knowledge crystal. Together they form a big knowledge crystal. It is possible that some items of the big knowledge crystal are contradicting each other, and there may be cases, which are not covered by any combination of the items. Assume that we have a job to produce a knowware from this crystal, covering all regulations that affect the life of all normal city habitants. Then we have to resort to a series of KM: namely those for knowledge selection, filtering, fusing, contradiction resolving, reorganization and so on. Not each of these jobs can be done automatically with some KM. Some manual work, in particular at current state of art of technology, is often unavoidable. Summarizing all these, we get a picture of a smelting furnace. The governmental regulations of all kinds are raw material of the furnace. The various kinds of knowledge processing on the crystal are the process of smelting. For this reason, we call it a smelting furnace model of knowware development as shown in Fig.1.

From the model of the smelting furnace we see also the importance of non-canonical knowledge processing that was mentioned at the beginning of this paper. The smelting process is not simple and straightforward. It deals with processing and fusing of incomplete, fuzzy, noisy and inconsistent knowledge. As a matter of fact, non-canonical knowledge processing is not something imposed on open and massive knowledge engineering. It is the latter's inherent and unavoidable property.

The model of developing knowware in the third example is different. The knowledge crystal is not built from a set of knowledge modules in a batch way, but is accumulated piece by piece and layer by layer from distributed knowledge sources. We call it a crystallization model of knowware development as shown in Fig.2. More details will be given with an example illustrated in the next subsection.

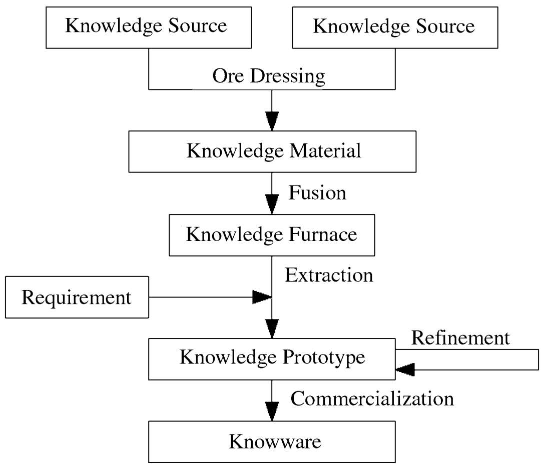

Fig.1. Furnace model.

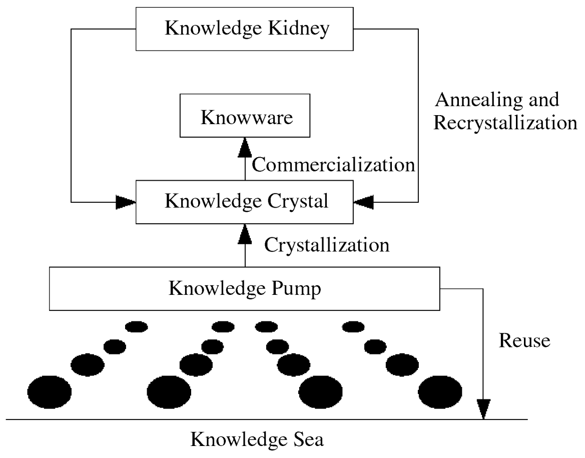

Fig.2. Crystallization model.

3.2 Knowware Construction with Crystallization Model

In this subsection, we will introduce a concrete example of knowware under development with crystallization model. The knowledge crystal to be grown, called KAU, is on knowledge about American universities. From this crystal, we want to develop a knowware, called KAUSA, of International Graduate Student Admission to American universities to help the young Chinese students to make their choice quickly when applying for a scholarship.

First, we need a knowledge distiller to collect information from the web pages of these universities. The basic technique we use is again PNLU. The PNL is called UKDL (University Knowledge Description Language) with a well designed grammar. The following are some sample sentential forms:

⟨applicants⟩ **must supply records of**
  ⟨course examination⟩ **plus**
  ⟨additional requirements⟩
⟨applicants⟩ **should take** ⟨course⟩ **exams**
  ⟨time⟩ **before** ⟨event⟩
⟨description⟩ **TOEFL score requested is**
  ⟨number⟩
⟨donner⟩ **provides scholarships to** ⟨receiver⟩
  from ⟨amount⟩ **to** ⟨amount⟩ **per** ⟨time⟩

where the bold faced words are keyword expressions of UKDL and those in brackets are parameters. The knowledge distiller is like fishing net consisting of these sentential forms, where the keyword expressions are head rope of the net and the parameters are the net nodes. This net is used to fish knowledge from Internet web pages about American universities. The denser the net nodes (i.e., the smaller the meshes) are, the more knowledge will be fished by the net. We use an ontological model to guide the fishing and to organize the fished knowledge. The structure of the ontology is like a semantic network or an ER diagram, where the nodes are agents and the edges connecting the nodes are relations between agents. More concretely, we use the Agent-Ontology representation developed in the Pangu project[13] for knowledge crystal KAU, which is a combination of agent (i.e., Capability-Belief-Strategy agent CBS agent for short) and ontological structure with CBS agents as basic nodes in the ontology. In this example, *university*, *applicant*, *record*, *TOEFL*, *scholarship*, *exam*, and so on are all agents; and supply, take, request, provide, and so on are all relations between agents. Each sentential form listed above with concrete parameters will be stored as a proposition in the corresponding agent. They are "half-finished product" of knowledge acquisition. In this way, an ontology is established for each university.

Now we come to knowware construction, i.e., KAUSA. As we said before, knowware is special purpose oriented. As an assistant to Chinese students, KAUSA should only contain knowledge of graduate student admission to American universities. Furthermore, the knowledge should be organized in a way that the students will have it easy to find any information needed for their application process. Therefore we use a Petri net like structure, different from the ontological structure used in KAU crystals. The new representation delineates the application process as a condition event net. All requests to the applicants are the preconditions, the fulfillment of requests is denoted with events, and the results of request fulfilled are the postconditions. For constructing KAUSA from KAU, a corresponding knowware specification should be prepared. This knowledge specification consists essentially of two parts: a list of constraints specifying which knowledge items to take, and a list of mappings specifying where these items should be mapped to.

Given that a knowware is generated from a crystal according to a specification, the generated knowware itself can be observed from different angles. Each such angle is called a user requirement. One of KAUSA's user requirements could be to "limit the search space to the top 10 American universities in computer science", or to "list the fitness indices of all universities for application given my TOEFL and GRE scores". The user requirements are met by knowware readers. The answers generated by knowledge readers are also in PNL form. We call this way of working PNLG (PNL Generation), which is the opposite of PNLU, and is also the fastest way of reproducing knowledge acquired with PNLU. Note that the knowware readers are developed separately by independent software engineers. For the same knowware, there may be many different knowware readers with different powers and different prices. After all, knowware and knowware readers are different commodities. This is also one of the reasons that the development of knowware and KU type knowledge middleware should be separated as two industrial branches. Another advantage of specifying user requirements is to make the knowware development fitting personal use. Note that the personal requirements of the students are quite different. An applicant in chemistry does not need to buy a KAUSA containing knowledge about biology faculties. By specifying customer's requirement, user-friendly knowware can be ordered and supplied for different private use easily.

4 Knowware in Information System Engineering

Since the last decade, information systems have been becoming larger and larger and are embedded more and more deeply in different organizational environments. The requirements of this kind of systems are in an organization-wide view. To obtain an organization-wide view involved many participants (from stakeholders to system developers). Many traditional information system development methods showed their limitations because they lack good manners for communicating between customers and system developers. The importance of integration and communication is obvious. Integration is one of the necessities for obtaining different views of the organization and establishing connections between these views. Communication between people ensures that the organization models are shared within the organization so that information can be used where it is relevant. An important way to achieve both effective integration and effective communication is to ensure that all parties involved have a shared understanding of the relevant aspects of an organization. In particular, when terms are used in a certain context, it must be clear what concept is being referred to.

Experience obviously shows that the system developers' knowledge on a particular domain may ease the tasks of information system development in this domain. The domain knowledge plays important role in the process of understanding the stakeholders' requirements and designing the satisfiable systems. Although the knowledge-based approach is not very new in information system engineering[14−17], it would been successful applied in a systematic manner by recent work on ontology[18,19]. Instead of only considering software development knowledge, we have paid special attentions to the representation and reuse of domain knowledge and built domain knowware for supporting the process of information system development. That is one of the distinguishing features of our approach compared with the other available efforts. In the rest of this section, we mainly illustrate the domain ontologies which have been used to facility the automatic information system engineering. These ontologies are in fact a series of domain knowware for information system engineering.

4.1 Organization Description Knowware

First of all, we introduce organization external ontologies as the meta-models for the organization business description so that customers can be guided to supply the relevant information. With these ontologies, mechanically understanding of these information becomes possible as the concepts in these ontologies have well-defined semantics and all the concepts in customers' business description are their instances and inherit their semantics. The top-level of these organization ontologies is shown in Fig.3 and the concept structure is in Fig.4.

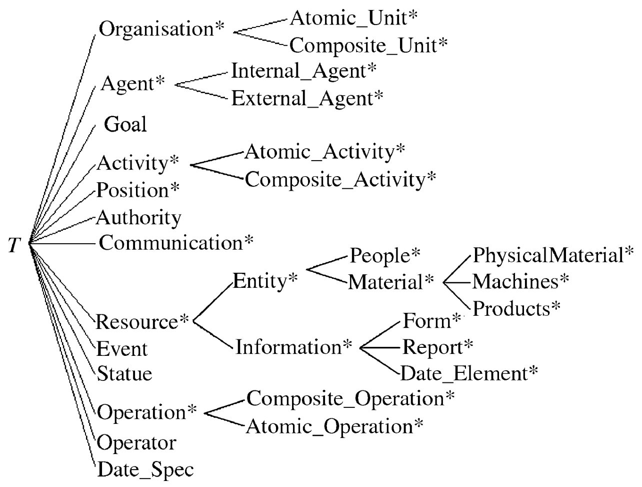

Fig.3. Top-level organization ontology.

With these ontologies, the process of information system requirements elicitation becomes a systematical process with the three features[20] as follows.

1) Customers can be guided to describe the business information in a systematic manner. In fact, a pseudo natural language (i.e., BIDL) has been designed for this purpose. The keywords in this language are the predefined associations of these ontologies and the terms the customers fill in the sentences are exactly the instances of the corresponding ontological concepts.

2) Analysts can easily understand the business descriptions with the pre-defined semantics of these ontologies. In fact the understanding could be automatic and the business information described supplied by customers can be automatically structured as concept graphs each of which is an instance of a part of organization concept structure.

3) The resulted concept graphs can be checked and validated for their completeness and consistency by the reference closure and the homomorphism between the expression of the business description and the corresponding knowware.

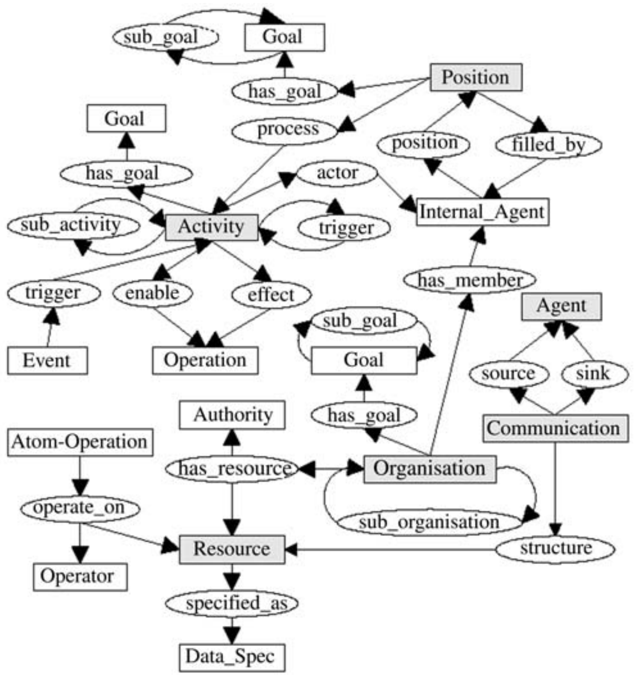

Fig.4. Organization concept structure.

4.2 Organization Modeling Knowware

Organization intensive ontologies contain some abstract models for modeling and analyzing the information system problem domains. At present, we consider three kinds of semantic models, i.e., the organization structure model, the organization goal model and the event/activity flow model.

Organization Model. This model makes the organization structure explicit. It models three kinds of associations, i.e., the *whole/part* association, the *leading* association and the *serving* association. The *whole/part* association between organization units characterizes the structural division of an organization and all of the *whole/part* associations form a hierarchy of organization units.

While the *whole/part* associations form an organization hierarchy of subordination relations, the *leading* associations constitute another hierarchy of leadership relations between organization positions. While modeling the organizations according to such a "leading" association, different organizations may have different shapes. Examples of common organization leading structures are hierarchical, matrix and flat.

Finally, the *serving* association means that an organization or position does something for another organization or position. No hierarchy exists for the *serving* associations. The structure of the organization structure ontologies could be formalized as:

Ontology OrganizationModel
  Concepts: Organization|Agent|Position
  Associations:
    whole-part: Organization$\rightarrow$Organization,
      Organization$\rightarrow$Agent
    leading: Position$\rightarrow$Position
    serving: Organization$\rightarrow$Position,
      Position$\rightarrow$Position,
      Position$\rightarrow$Organization

Organization Goal Model. This model tries to capture various semantic links between organization goals. There are several kinds of semantic links:

- a goal and-reduction means that a goal could be reduced into a conjunction of a set of its sub-goals;
- a goal or-reduction means that a goal could be reduced into a disjunction of a set of its sub-goals. In the goal and-reduction and the goal or-reduction, the goal reduction is in fact the goal assignment;
- the mean-ends link means an organization goal could be accomplished by an activity goal;
- the goal dependence means that a goal depends on the dependee, i.e., the achievement of the dependee is the prerequisite for achieving this goal;
- the goal conflict means that a goal may interfere in the interferee, i.e., it prevents from the achievement of the interferee.

The goal model is represented as:

Ontology GoalModel
  Concepts: Goal(:= OrganizationGoal | RoleGoal) |Activity
  Associations:
    andReduction: Goal$\rightarrow \mathbb{P}$Goal
    orReduction: Goal$\rightarrow \mathbb{P}$Goal
    conflict: Goal$\rightarrow$Goal
    depend: Goal$\rightarrow$Goal
    assignment: OrganizationGoal$\rightarrow$RoleGoal
    meanEnds: RoleGoal$\rightarrow$Activity

Event/Activity Model. This model is for modeling the flows of events and activities in an organization. Here, we assume that any activity is activated by an event and has an enable state (enable) and results in another state (effect), which may inversely cause other events. Suppose that we use concept Connector= $\{\vee, \wedge, \neg, =\}$ to represent the connectors between states and between state and activity. Then we have the following formulation:

```
Ontology EventActivityModel
  Concepts: Event|State|Activity|Connector
  Associations:
    flow: State→Activity, Activity→State,
          State→Connector, Connector→State,
          Activity→Connector, Connector→Activity,
          Connector→Connector
    trigger: Event→Activity
    occur: State→Event
```

Notice that all of the concepts of *Organization*, *Position*, *Goal*, *Agent*, *Event*, *State* and *Activity* are coming from the expression of the business description. With these knowware, the process of organization modeling and analysis can be automatically completed by using ontological graph reconstruction algorithms[21].

4.3 Information System Modelling Knowware

Having got the organization models, the next step of information system engineering is information system modeling. Some knowware have been developed to facilitate the construction of these models.

Information system model refers to roles, activities and data. These basic types are the role objects, the activity objects and the data objects. Each role object corresponds to an organization concept, such as an organization, an agent, or a position. The features of the role objects include that each of them can perform some activities and has visiting authority to some data. Each activity object is a data processing procedure which has a set of input data, a set of output data and a set of functional mappings from input data to output data. Each data object has a set of attributes describing its structure. The information system modeling knowware is represented as:

```
Ontology InformationSystemModel
  Concepts: RoleObject|DataObject|ActivityObject
  Associations:
    assignment: RoleObject→ActivityObject,
    authority: RoleObject→DataObject×AUTH ①,
    input,output: ActivityObject→DataObject,
    leading: RoleObject→RoleObject,
    subConcept: DataObject→DataObject,
            ActivityObject→ActivityObject,
    entityRelation: DataObject→DataObject
```

In which, each *RoleObject* comes from an instance of an organization concept and a position/agent concept. each *DataObject* comes from an instance of a resource concept. And each *ActivityObject* comes from an instance of an activity concept.

4.4 Summary on Information System Knowware

Above knowware are some of the domain knowware special for the organization information systems. Other knowware also include knowware of software development, such as the knowware on software architecture as well as those on the application implementation (please refer to [10] for detailed description).

With these knowware, the information system development becomes a process knowware-driven information extraction and model reconstruction. This process might be iterative and of step-wise refinement. The whole framework can be depicted as shown in Fig.5.

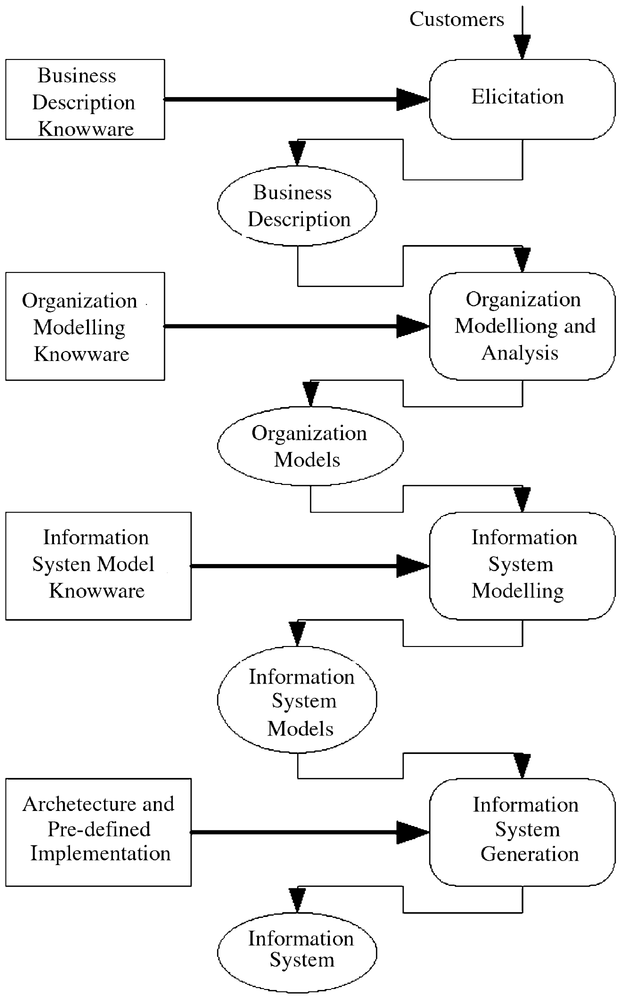
Customers
Business Description Knowware
Elicitation
Business Description
Organization Modelling Knowware
Organization Modelliong and Analysis
Organization Models
Information Systen Model Knowware
Information System Modelling
Information System Models
Archetecture and Pre-defined Implementation
Information System Generation
Information System

Fig.5. Knowware-driven information system engineering.

5 Conclusions

Knowware is proposed for separating the knowledge component from software and making knowware and software two different research topics and salable commodities. When they could be delimited with a clear boundary, a client could buy software tools from a software developer and knowware from a knowledge developer. In this sense, knowware and knowware engineering call forth a new industry, i.e., the knowledge industry. For growing this new industry, many issues should be further addressed which include methodologies, techniques and tools of knowware development. The comparative study with the software development is worthwhile. And from our experience until now, ontologies seem to be a solution as the basic knowware structures.

① AUTH={RO, WO, RW}, which stand for read-only, write-only and read-write respectively.

If that is true, the ontology relevant theories and techniques should be systematically studied and a unified knowledge modeling framework will be very appealing. By the way, a variety of different domain knowware should be developed and make their own contributions to different applications.

Acknowledgements The authors are very thankful to Profs. Fei-Yue Wang, Hong Mei, Jian Lü, Xiao-Guang Yang and Dr. Lin Liu for their fruitful discussions on knowware. The authors also thank the anonymous referees for their valuable comments, which have helped to improve the manuscript.

References

[1] Feigenbaum E A. Some challenges and grand challenges for computational intelligence. *Journal of ACM*, 2003, 50(1): 32–40.

[2] Lenat D B, Guha P V. Building Large Knowledge Based Systems: Representation and Inference in the CYC Project. Addison Wesley, 1990.

[3] Cungen Cao *et al.* Progress of national knowledge infrastructure. *Journal of Computer Science and Technology*, 2002, 17(5): 523–534.

[4] Yolanda Gil, Enrico Motta, V Richard Benjamins, Mark A Musen (eds.). The Semantic Web—ISWC 2005: 4th International Semantic Web Conference. *Lecture Notes in Computer Science 3729*, Berlin/Heidelberg: Springer, 2005.

[5] Cannataro M, Talia D. Knowledge grid: An architecture for distributed knowledge discovery. *Communications of the ACM*, January 2003, 46(1): 89–93.

[6] Lois M L Delcambre, Christian Kop, Heinrich C Mayr *et al.* (eds.) Conceptual Modeling — ER 2005, 24th International Conference on Conceptual Modeling. *Lecture Notes in Computer Science 3716*, Berlin/Heidelberg: Springer, 2005.

[7] Ruqian Lu. Pseudo-Natural Language Understanding and Knowledge Acquisition. Some Important Issues of Chinese Information Processing, Bo Xu, Maosong Sun, Guangjin Jin (eds.), Science Press, 2003, pp.229–245.

[8] Ruqian Lu, Cungen Cao. Towards Knowledge Acquisition From Domain Books. Current Trends in Knowledge Acquisition, IOC, Amsterdam, 1990, pp.289–301.

[9] Ruqian Lu, Cungen Cao, Yonghong Chen, Zhangang Han. On automatic generation of intelligent tutoring systems. In *Proc. 7th Int. Conf. AI in Education*, 1995, pp.67–74.

[10] Ruqian Lu, Zhi Jin. Domain Modeling Based Software Engineering: A Formal Approach. Kluwer Academic Publisher, 2000.

[11] Ruqian Lu. From hardware to software to knowware: IT's third liberation? *IEEE Intelligent Systems*, March/April 2005, pp.82–85.

[12] Feigenbaum E A, McCorduck P. The Fifth Generation, Artificial Intelligence and Japan's Challenge to the World. Addison-Wesley, 1983.

[13] Ru-Qian Lu *et al.* Agent-oriented commonsense knowledge base. *Science in China* (Series E), 2000, 43(6): 641–652.

[14] Loucopoulos P, Champion R E M. Knowledge-based support for requirements engineering. *Journal of Information and Software Technology*, 1989, 31(3): 123–135.

[15] Mylopoulos J, Borgida A, Jarke M, Koubarakis M. Representing knowledge about information systems. *ACM Trans. Office Information Systems*, 1990, 8(4): 325–389.

[16] Lu R, Jin Z, Wan R. PROMIS: A knowledge-based tool for automatically prototyping management information systems. In *Proc. AVION'94*, Paris, France, 1994, pp.325–330.

[17] Sutcliffe A, Maiden N. The domain theory for requirements engineering. *IEEE Trans. Software Engineering,* 1998, 24(3): pp.760–773.

[18] 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.

[19] Uschold M, King M, Moralee S, Zorgios Y. The enterprise ontology. *The Knowledge Engineering Review*, 1998, 13(1): 31–89.

[20] Zhi Jin, David A Bell, F G Wilkie, D G Leahy. Automated requirements elicitation: Combining a model-driven approach with concept reuse. *International Journal of Software Engineering and Knowledge Engineering,* 2003, 13(1): 53–82.

[21] Zhi Jin, Ruqian Lu, David A Bell. Automatically multi-paradigm requirements modeling and analyzing: An ontology-based approach. *Science in China* (Series F), 2003, 46(4): 279–297.

Ru-Qian Lu is a professor of computer science of the Institute of Mathematics, Academia Sinica. His research interests include artificial intelligence, knowledge engineering and knowledge based software engineering. He has won two first class awards form the Academia Sinica and a National second class prize from the Ministry of Science and Technology. He has also won the sixth Huo Lookeng Prize for Mathematics.

Zhi Jin was awarded a B.S. degree in computer science from Zhangjiang University in 1984, and studied for her M.S. degree in computer science (expert system) and her Ph.D. degree in computer science (artificial intelligence) at Changsha Institute of Technology. She was awarded the Ph.D. degree in 1992. She is a senior member of China Computer Federation. Her research interests include knowledge-based systems, artificial intelligence, requirements engineering, ontology engineering, etc. Her current research focuses on ontology-based requirements elicitation and analysis. She has got about 80 publications, including co-authoring one book.