An Approach to Automatic KBS Construction From Reusable Domain-specific Components

Yasser Abdelhamid, Hesham Hassan, Ahmed Rafea
Central Laboratory for Agricultural Expert Systems
{Yasser, Hesham, Rafea} @esic.claes.sci.eg

Abstract

The demand for high quality, large scale knowledge based systems has increased to the point where great improvements in knowledge engineering technology are needed. Library based approach for knowledge reuse is one of the widely exploited approaches, but there are two obstacles to this approach: the indexing problem, and the configuration problem. The real cause of these problems is the high level of abstraction employed to the library components. In this paper, we propose an approach to overcome these problems by building domain-specific knowledge components libraries, and automating the process of building knowledge based systems.

I Introduction

Many approaches have been exploited for accelerating the process of building KB systems, some of these approaches were concentrating on the portability of ontologies, so that it can be reused over a wide range of domains e.g. KACTUS [1]. Some other approaches were focusing on constructing a library of problem solving methods, that is somehow, generic enough to be reused in different applications e.g. Generic Tasks [2,3], Components of Expertise [4], CommonKADS [5]. Another approach was to build a complete framework for building knowledge based systems from generic library components e.g. PROTEGE-II[6,7].

Reusability is not a new term in the field of software engineering, it has been used long ago to promote the productivity of software developers. The same approach has been applied to knowledge engineering, since KBS developers often duplicate work on similar systems.

The cost of building software from reusable components should not exceed that of building the system from scratch. Actually the cost of building a software system from reusable components is the cost of locating the most appropriate reusable software component for the current application, according to certain criteria that is application dependent, and the cost of fitting selected components together so that the output of one component matches the input of another. Knowledge engineering researchers used to refer to the former by the

refer to the latter by the

To build a library of reusable components, one should decide on what is the right granule size of a component. There are three criteria that can be used in evaluating the efficiency of a library, the component reusability range, the component selection cost,

and the configuration cost. On one hand, as the granule size of a reusable component grows, its reusability range shrinks, the selection cost, and the configuration cost decreases. On the other hand, as the component size becomes smaller, the range of its reuse becomes wider, but both the selection cost and the configuration cost becomes higher.

This work addresses the possibility, and feasibility of building knowledge based systems out of reusable, domain-specific library of knowledge components. For convenience, we shall stick to the terminology provided by CommonKADS [4] for referring to different types of knowledge.

IIAnatomy of KBSs

There are many knowledge component types that constitute a KBS. As defined by CommonKADS, there are three types of knowledge: domain knowledge, inference knowledge, and task knowledge.

Fig. 1 displays the internal structure of a KBS, and the interdependencies between each of its components.

Domain knowledge is structured internally into two layers: domain ontology layer, and domain models layer. Domain ontology layer is the most fundamental part of a KBS, and it includes domain concepts declarations, and domain relations which are built on top of the declared concepts, in the form of rules, tables, mathematical functions, object hierarchies, constraints, or any other representation. Domain models are built on top of domain relations, each domain model points to a selected set of domain relations, which together represent the function defined by the domain model.

Inferences are built on top of the domain knowledge layer. The basic constituents of an inference are input roles, and output roles. Input roles are classified internally into dynamic input

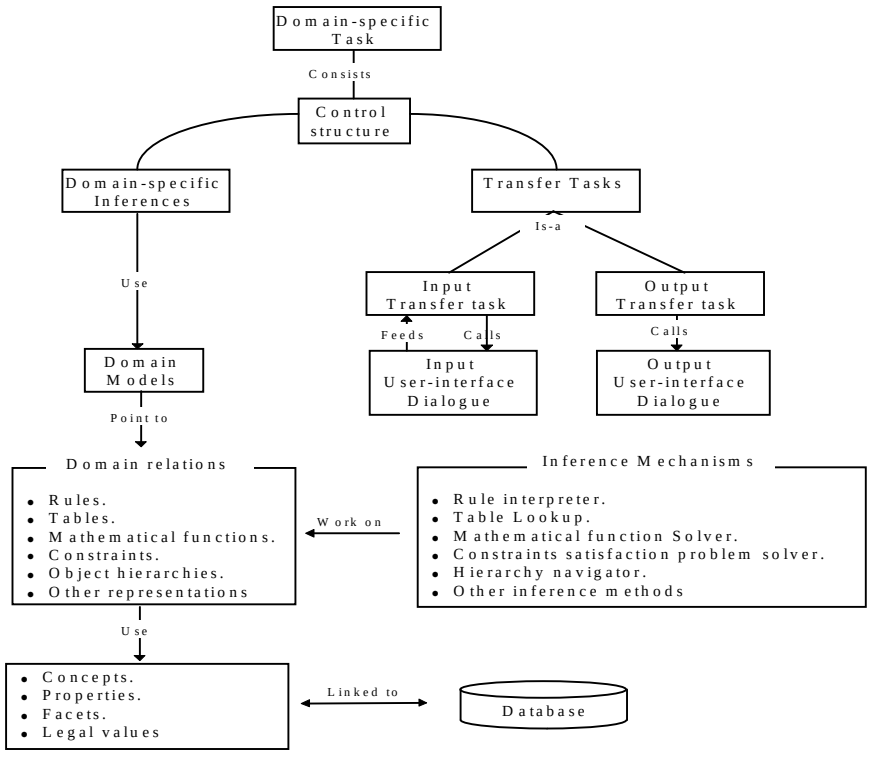

Fig.1 Types of knowledge

roles, and static input roles. Static input roles are merely domain models that are used to derive values of output roles, given the values of dynamic input roles.

Tasks are control structures over transfer tasks and inferences. Transfer tasks are either input transfer tasks or output transfer tasks. Input transfer tasks call user interface modules that acquire input from the user, and update the working memory with the acquired data. Output transfer tasks, send output data to output user interface modules that display the output to the user in the appropriate form.

External agents like databases, are interfaced through the declaration of its structure in the knowledge base.

III Reusability

This section discusses different levels of reusability that one KBS component type can provide, then we investigate each component type with regard to its reusability capabilities.

III-1 Reusability Levels

To talk about knowledge reuse we should first put highlights on what to be reused, and then we should specify how this reuse would take place. Not all types of knowledge are reusable at the same level, there is always a limitation on the reusability range of a knowledge

beyond this range. This range of reusability depends on the knowledge components nature,

As illustrated in Fig.2, we can distinguish four levels of reusability, according to the range that one knowledge component is limited to. A knowledge component is either a generic component, so that it can be reused in different domains of applications, and different task types, without any restrictions, or it is restricted to a specific range, and it

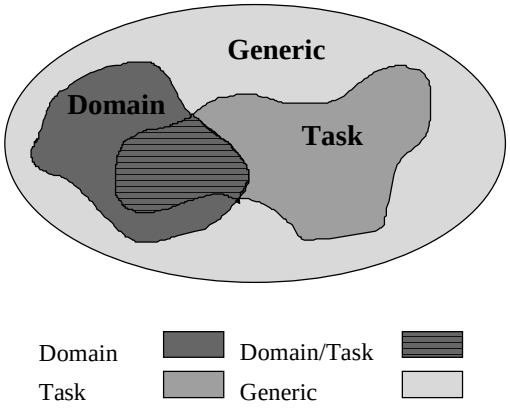

Fig.2 Levels of reusability

One knowledge component, is either restricted to a specific domain, a specific task, or both domain and task. If it is domain restricted, then it can be reused over all applications in that domain. If it is task restricted, then it can be reused across all domains of applications, but in the same task. If it is both, domain and task restricted, then it can only be reused in a specific domain, and a specific task, and this is the least level of reusability.

III-2 Reusability Competence Of KBS Component Types

As illustrated in Fig.1, we can construct reusable libraries of the following knowledge component types:

- Tasks.
- Inferences.
- Inference mechanisms.
- Domain knowledge (Ontology and Domain models)

Task library: Is a collection of abstract skeletal plans that exhibit the overall behavior of a task type. A task model constitutes the procedural knowledge of a task, that is how the expert system

would behave to fulfill the required function.

Task models like diagnosis, and scheduling are not domain restricted knowledge components, since they can be reused over different domains, but they are task restricted because they can only be used to fullfill a specific task type.

Inference library: The main function of an inference is to expand the set of known values of concepts properties, going from known facts or hypothesis to conclusions using the appropriate knowledge supplied by domain models. Inferences maintain its reusability from its ability to use different domain models as its static roles. The same inference can be used in different tasks by altering the domain model to suit the application on hand. They are neither task-specific, nor domain-specific, because one inference,

tasks, and different domains.

Inference mechanisms: Which have a very flexible range of application, since each inference mechanism has its own knowledge representation that it works on, and it has nothing to do with the content of knowledge, neither the role that it plays in the task structrue. It is only restricted to knowledge representation, so, we can conclude that inference mechanisms are fully generic components.

Knowledge base: Which is the fundamental constituent of a knowledge based system. As mentioned before, we can organize the knowledge base into two layers, domain ontology, and domain models. Domain ontology represent the communication language of all components that constitute the expert system, accordingly, the reusability range of ontology is over all tasks within the same domain.

Domain models represent clusters of domain relations used for the achievement of a specific function. Their use is limited to their domain of application.

IV Suggested Approach

The library driven approach is one of the promising approaches to knowledge reuse. Unfortunately, there are two inherent problems that elevate the cost of building a KBS from reusable library components, namely: indexing problem, and configuration problem. Knowledge engineers are used to spend a considerable effort to locate the appropriate knowledge components that suit the requirements of the intended application. This problem arises because generic library components are usually designed at a high level of abstraction, that makes it after some customization processes, suitable for applications instances. This customization process (or configuration process) varies from one domain of application to another, and usually requires a great effort from the knowledge engineer. By all means, the effort spent in locating the right component, and configuring the selected component should not exceed the required effort for building the application from scratch.

Our approach to this problem is twofold. We build a customized (pre-configured) library of knowledge components for a specific domain, this is the first half of our solution. This helps in two ways: First, we reduce the number of library components to those only applicable for the domain of interest. Second, as we already know in advance the requirements of

applications in the selected domain, we can customize library components for the desired applications instances automatically, and this is the second half of the solution.

According to this solution, we are expecting the following advantages:

1. The cost of selecting and configuring reusable components can be minimized by utilizing reusable domain-specific knowledge components libraries, and automatic configuration tools.
2. From the knowledge engineering perspective, building a KBS from reusable domain-specific task models guides both knowledge acquisition and KBS development processes in a structured manner.
3. A knowledge based system, as a software, has many routine, and mundane software engineering tasks that can be reliefed by utilizing automatic code generation facilities.

V Expert System Development Environment

The proposed development environment consists of a set of editors, that facilitate building and maintaining knowledge components libraries, and constructing knowledge based systems out of these libraries.

As illustrated in Fig.3, the proposed development environment consists of a set of tools that enable knowledge engineers to build libraries of different types of reusable knowledge components, and to build application instances from these components.
To build the infrastructure for a large scale knowledge based system, we start with acquiring ontology, that is the most fundamental constituent of a knowledge base. By ontology, we mean a full declaration of concepts, relations between concepts, and relations between properties.

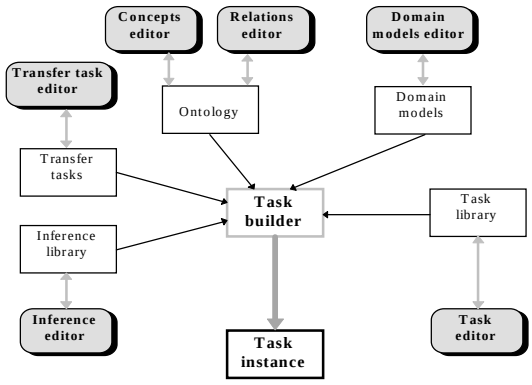

Fig. 3 Integrated expert system development environment

This part of ontology represents the lower layer of a domain knowledge. Domain models embodies the upper layer, and represent knowledge of what are the domain relations required to fulfill a specific function within the domain of application.

To acquire ontology, we prepared two editors: Concepts editor, and Relations editor. The concepts editor is used to declare domain concepts, along with their properties, and properties facets. The relations editor, uses the concepts declarations as input, and facilitates defining relations between concepts, and relations between properties that

Currently we have five distinct representations of relations: Rules, Tables, Mathematical functions, Constraints, and Object hierarchies. Each of these representations has its own internal structure that suits different kinds of domain relations. The selection of the appropriate representation of a specific domain relation is up to knowledge engineer, according to some criteria that characterizes each representation. It is noteworthy that the proposed

archetecture is not closed to the aforementioned knowledge representations, the knowledge engineer is free to add new knowledge representations, and inference mechanisms that work on them.

After constructing concepts and relations, we define domain models using domain models editor. Domain models editor creates clusters of relations drawn from the knowledge base, these relations are used together to fulfill a specific function at the knowledge level.

Task editor is used to create skeletal task structures that embody the procedural knowledge of a task, possible entries of a task model are:

- Initialization entries.
- Transfer tasks calls.
- Domain inferences calls.
- Control entries, like loops and conditional statements.

Transfer tasks editor creates input transfer tasks, and output transfer tasks, and user interface modules. User interface modules are responsible for accessing and displaying data in the appropriate form to enhance the usability and acceptability of the KBS, while transfer tasks collect, validate and communicate data to, and from user interface modules.

Inference editor creates prototypical domain-specific inferences, they are represented as frames of code in which the variable parts can be filled with domain models instances.

The task builder comes at the task instance generation stage, where the knowledge engineer selects the appropriate task model from the task models library, and the task builder takes the selected model as input, and interactively creates task instance by asking the knowledge engineer for transfer tasks and inferences to be used in the generated task instance.

Each transfer task is hooked to a user interface module, and the knowledge engineer only selects the transfer task to be used in the current task, from the transfer tasks that have been developed using the transfer tasks editor.
When the task builder encounters an inference call in the task model, it retrieves the inference prototype from inferences library, then, it starts asking the user for actual domain models to be inserted into the inference frame. By the end of this process, the inference prototype becomes completely instanciated. After the task builder instanciates all task model entries, the task model becomes complete.

VI Building Knowledge Based Systems

After we have introduced the proposed KBS development environment, in this section we describe how we build applications instances using this environment.

The Central Lab for Agricultural Expert Systems has succeeded in building a library of knowledge components in the domain of agriculture, as we have a considerable experience with applications in this domain, gained through building a number of expert systems, we constructed the following libraries:[8]

1. **Task models library:**
This library contains an abstract model for different applications in the domain of agriculture, actually we have five

```
Begin
    Initialize the system.
   Get case data ( a transfer task ).
    Abstract for irrigation ( inference ).
    Compute irrigation interval ( inference ).
    Generate initial irrigation schedule ( inference ).
    WHILE TRUE
        Check violations ( inference ).
        IF  ACCEPTED
            Exit
        ELSE
            Revise irrigation schedule ( inference ).
        ENDIF
    ENDDO
    Display irrigation schedule ( a transfer task ).
End
```

Fig.(4) Irrigation Task Model

task types: irrigation, fertilization, diagnosis, treatment, and plant care.

2. **Inference library:**
This library contains a collection of domain inferences, from which an application instance is constructed.

3. **Domain models library:**
This library contains a definition of different domain models used by inferences as static roles. These domain models point to different sets of domain relations defined by the domain ontology.

4. **Domain ontology:**
A repository of domain-specific concept declarations, and domain relations, which represent the core of knowledge acquired through different implemented applications.
The following is an example of how we build an irrigation application instance using the task builder.

First we select the task model from the task models library. Note that model selection is very simple, since we already know the application model that we need. Fig. 4 illustrates the task structure of the selected model.

The inference structure of the irrigation model is illustrated in Fig.5, and displays the different domain inferences and domain models used by the model.

As shown in Fig.5, the inference structure of the irrigation model uses four domain models, namely: derivation model, irrigation model, constraint model, and fix model.

Derivation model uses knowledge required to derive irrigation parameters needed by the system, like plant growth stage.

Irrigation model constitutes knowledge required to calculate the irrigation interval and water quantity for the current case.

Constraints are conditions that must be satisfied to accept the generated irrigation schedule.

Fix model represents knowledge required to modify the generated irrigation schedule to satisfy the conditions specified in the constraints model.

The task builder is an interactive tool that allows the knowledge engineer to select the task model from which an instance is created for the current application. The knowledge engineer can alter the inferences of the selected task model if needed. The task builder takes the selected model as a plan, and starts to ask the knowledge engineer for the actual domain models to be used by the task. These domain models are drawn from the pre-constructed domain models library, also the knowledge engineer can modify the relations set that constitute a selected domain model, either by selecting other relations from the domain ontology library, or by acquiring new specific knowledge for the current application.

Eventually, the task builder generates the task instance code, taking in

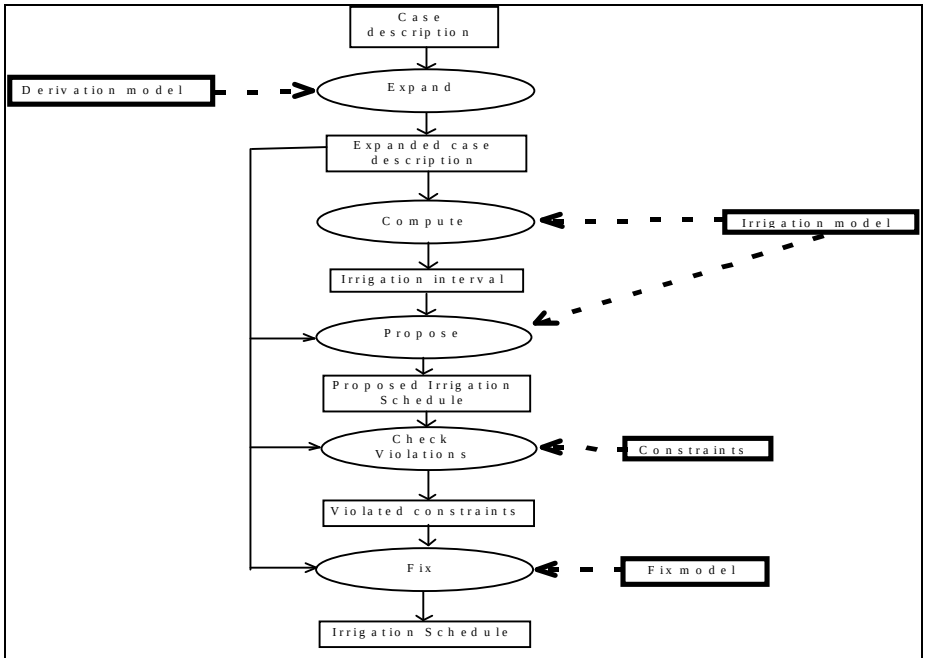

Fig.(5) Irrigation Inference Structure

consideration all the integrity of all inferences that constitute the task.

VII Conclusion

In this paper we were focusing on the problems of building large scale KBSs from reusable components. We introduced the reusable libraries technique as an accepted, and promising solution, and we described the indexing problem, and the configuration problem as the main cost items that guide the decision whether building a KBS from reusable knowledge components libraries is feasible or not.

We investigated the reusability opportunities of KBSs components, and we studied the behavior of each type when prepared for reuse, and suggested the proper way of utilizing these opportunities.

We presented domain-specific libraries and automatic configuration as a solution for the forementioned problems. Domain-specific libraries narrows the selection range to the scope of interest, therefore it relieves the effect of indexing problem, on the other hand, automatic configuration guides the process of building customized applications from predefined skeletal or abstract structures, taking care of system integrity and robustness.

We introduced a suite of editors and modules that build up an integrated KBS development environment, based on the utilization of domain-specific libraries of reusable knowledge components, and automatic configuration of custom KBSs.

The paper also included an illustrative example that shows how a task instance can be constructed from reusable knowledge components, using the task builder tool.

VIII References:

[1] B. J. Wielinga and A. Th. Schreiber. Conceptual modelling of large reusable knowledge bases. In K. von Luck and H. Marburger, editors, Management

and Processing of Complex Data Structures, volume 777 of Lecture Notes in Computer Science, pages 181-200, Berlin, Germany,. Springer Verlag, 1994.

[2] B. Chandrasekaran, Generic Tasks as building blocks for knowledge based systems: The diagnosis and Design examples. Knowledge engineering review,4,183-219,1987.

[3] B. Chandrasekaran, Towards a taxonomy of problem solving types. AI Magazine, 4(1),9-17,1983.

[4] L. Steels Components of expertise. AI Magazine, 11(2):29-49, 1990.

[5] B.J. Wielinga, W. Van de Velde, A. Th. Schreiber, and J.M.Akkermans.

modeling approaches
David, Jean-Paul Krivine, and Reid Simmons, editors, Second Generation Expert Systems, Pages 299-335, Springer-Verlag, Berlin - Heidelberg, Germany, 1993.

[6] Puerta, A.R., Edgar, J.W., Tu, S.W., and Musen, M.A.: A multiple method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. Knowledge Acquisition, 4, 171-196, 1992.

[7] Puerta, A.R., Tu, S.W., and Musen, M.A.: Modeling tasks with mechanisms. International journal of intelligent systems,8,129-152,1993.

[8] Technical Report No. TR-88-024-41, Methodology for the Engineering of Expert Systems - Version 6.0, December, 1995.