Conceptual Modelling for Knowledge-Based Systems

P-H. Speel [1], A. Th. Schreiber [2], W. van Joolingen [2], G. van Heijst [3], G.J. Beijer [4]

[1] Unilever Research Vlaardingen, [2] University of Amsterdam, [3] CIBIT, [4] Bolesian

1 INTRODUCTION

1.1 Kowledge-based systems

A knowledge-based system (KBS) is a software system that contains a significant amount of knowledge in an explicit, declarative form. The area of KBS development has matured over the past two decades. It started with first-generation expert systems with a single flat knowledge base and a general reasoning engine, typically built in a rapid-prototyping fashion. This has now been replaced by methodological approaches that have many similarities with general software-engineering practice. KBS development is best seen as software engineering for a particular class of application problems. These applications problems typically require some form of reasoning to produce the required results. In current business practice there is an increasing need for such systems, due to progression of information technology in our daily work. Some typical applications are systems for assessing loans in a bank, for job-shop scheduling in a factory, for configuring an elevator, and for diagnosing problems in a production line.

In Table 1 we have listed some terms that are frequently used in the area of KBS development. Please note that in practice it is difficult to define a strict borderline between information and knowledge. For this reason there can also not be a strict separation between KBSs and other software. There is a gradual scale of knowledge-intensiveness of application problems, and a KBS system is typically chosen if the application is on one side of the spectrum.

1.2 Methodologies for KBS construction

Similar to other software systems, a methodological approach is essential for KBS development. The days are long gone that expert systems were developed with just rapid prototyping. These systems do not scale up and cannot be maintained. Over the past decade a number of specialized methodologies have been developed for this field. Some well-known methodologies are CommonKADS (Schreiber et al. 2000), PROTÉGÉ (Tu et al. 1995), and MIKE (Angele et al. 1998). In this article we mainly base ourselves on the CommonKADS approach.

The construction of a KBS poses some specific problems. A central problem that needs to be tackled by a methodology for KBS development is the acquisition and representation of knowledge. This activity is usually called knowledge engineering, although some people use this term for the whole process of KBS construction.

Term	Description
Data	The raw digital material, e.g. a Morse signal "…---…"
Information	Interpreted data, e.g., the message "SOS"
Knowledge	Assigns a purpose and/or action to information, e.g., "SOS" is an emergency signal that requires an immediate response in the form of a rescue action.
Knowledge-based system	A software system with an explicit, declarative description of knowledge for a certain application.
Knowledge system	Synonym for knowledge-based system.
Expert system	Old term for knowledge-based system. Should not be used because it may give rise to wrong expectations. In general, a knowledge-based system is not intended to replace an expert but to support an expert.
Domain	Some area of interest, e.g. a production line, a medical discipline, banking services.
Task	A piece of work that needs to be done. Some typical tasks encountered are diagnosis, assessment, and planning. A business process is composed out of a collection of tasks to achieve a business objective.
Agent	An executor of a task, typically either a human or a software system.
Application	The context in which an agent executes a task in a particular domain.

Table 1: Terminology in the KBS field

Before we move on, we have to consider what constitutes a methodology. A methodology always has a number of building blocks or elements. These elements take the form a pyramid (see Figure 1). The cornerstone of a methodology is its worldview. Object-orientation is an example of a worldview. For KBS construction the worldview is typically related to the general approach of knowledge engineering, and how it is intertwined with the other software-engineering activities. The worldview takes shape in a theory, that describes the basic concepts and models of the approach. In the rest of this section we discuss the worldview and theory in some more detail. In the rest of the article we dive deeper into the theory and also discuss supporting methods en techniques.
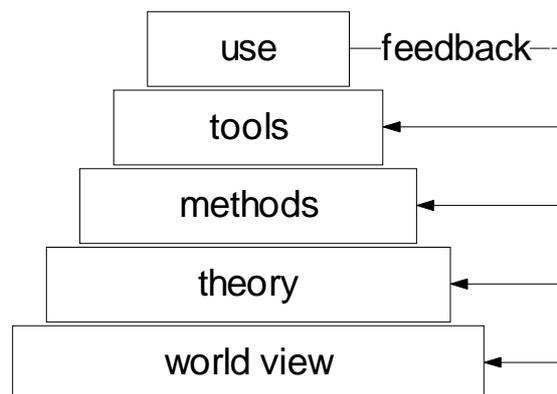

Figure 1: The methodological pyramid

1.3 Knowledge engineering as a modelling activity

Experience has shown that eliciting and explicating knowledge is best seen as a modelling activity, also called conceptual modelling. This activity takes the form of a specialized type of

requirements engineering. KBS construction methods typically provide tools for knowledge analysis in the form of so-called conceptual models of knowledge or simply knowledge models. A knowledge model provides an implementation-independent specification of knowledge in an application domain. Typically, a knowledge model provides formats for writing down both static domain knowledge (rules, classes, relations) as well as reasoning strategies in which this domain knowledge is used to solve a particular problem. An important feature of knowledge-engineering research is that it provides us with predefined, reusable models for certain knowledge-intensive tasks, also called problem-solving methods (Benjamins and Fensel 1998; Motta 1999; Breuker and Van de Velde 1994; Hayes-Roth et al. 1985; Clancey 1985). Figure 2 shows the typology of knowledge-intensive tasks used in the CommonKADS approach (similar typologies are used in other approaches). The typology distinguishes analytic and synthetic tasks. Diagnosis and assessment are typical examples of analytic tasks. In assessment the goal is to take a decision based on data about a case and a set of norms or criteria. An example of assessment is the assessment of a mortgage application by a bank. Diagnosis is concerned with finding the fault that causes a device or biological system to malfunction. The underlying knowledge typically contains a behavioral model of the system under consideration. An example of a synthetic task is configuration design, in which an artifact is constructed from a set of predefined components. The resulting design should satisfy a set of constrains and preferences.

For each task type knowledge models can be found in the literature (Benjamins 1993, Breuker and Van de Velde 1994, Puppe 1993, Benjamins et al. 1996, Motta 1999). The idea is that the system analyst decomposes the application task to the level where she can identify such task types. At that point an existing knowledge model is selected and "plugged in." In Section 'Conceptual Modelling' we show an example of this type of conceptual modelling. The chosen model often needs some fine tuning and needs to be extended with application-specific knowledge. Industrial applications of diagnosis models are available (Speel and Aben 1997; Speel and Aben 1998; Orsvärn 1996a; Orsvärn 1996b).
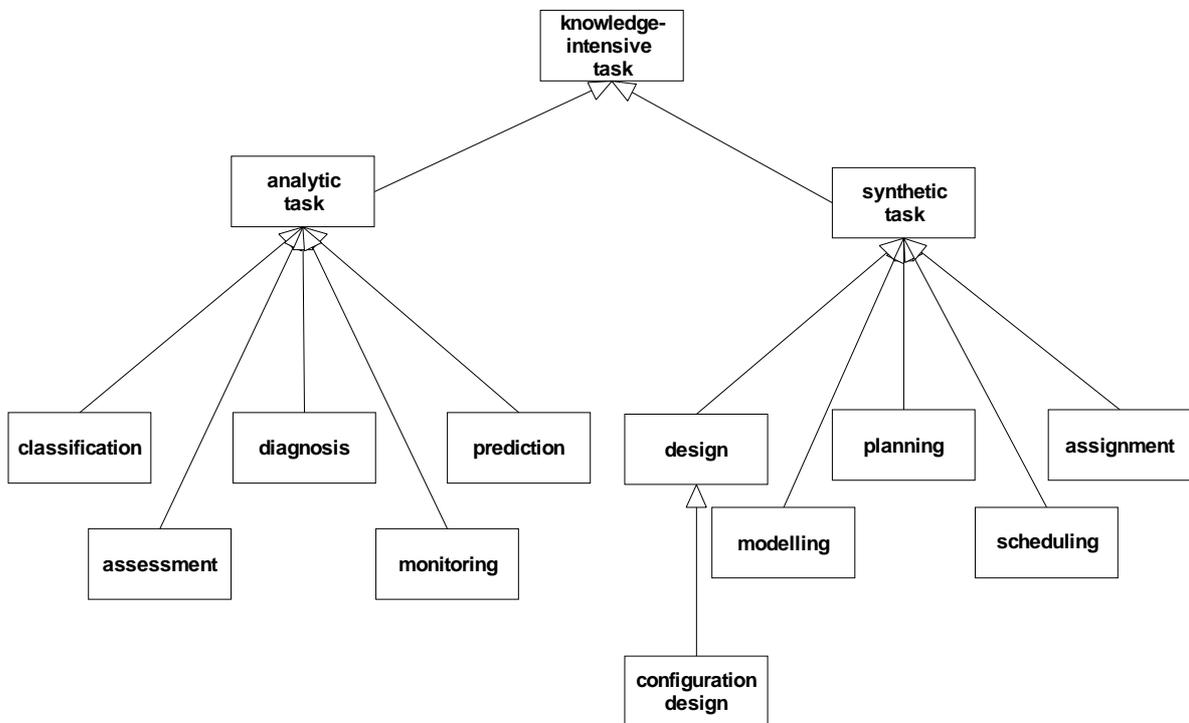

Figure 2: Typology of knowledge-intensive task types (from Schreiber et al., 2000)

1.4   Business modelling and the process of KBS construction

Conceptual modelling is a central activity in KBS construction, but it of course only part of the story. Knowledge technology has suffered a lot from technology-push approaches, in which a system was forced into an organization. This does not work, especially not for knowledge-intensive tasks, because partial or full automation of such tasks has many implications for the business process, the people involved, the quality of work, and so on. This means that conceptual modelling always needs to be proceeded by a careful business analysis, in which opportunities for KBS construction are identified, and accompanying measures are prepared to ensure that the KBS solution is accepted by the organization. The CommonKADS methodology provides a set of worksheets to support business modelling for KBS construction (Schreiber et al. 2000, Chapter 3).

Because of the specific role of conceptual modelling, communication aspects are usually separated out in KBS construction. This means that the system analyst builds a separate specification of user-system and system-system interactions. The rationale behind this division of work is a "divide & conquer" approach: during conceptual modelling you do not have to worry about external interaction and vice versa. Communication modelling in KBS construction is in essence not very different from "standard" communication modelling, although there may be some specific requirements. An example of the latter is the need for an explanation facility that can be used to explain to a user the rationale behind a reasoning process.

Together, business modelling, conceptual modelling, and communication modelling provide the requirements for building the KBS software. KBS design and implementation is not inherently different from general software design. Typically, you might want to use some KBS-specific software environments, although this is not a prerequisite. A KBS is hardly ever a "one-shot" system. Exploratory prototypes may be required to find out whether the system behaves in the manner expected by the users and domain specialists. For this reason, much emphasis is placed on KBS design on clear, almost transformational, routes from analysis to design. This is in line with modern developments in object-oriented system development (see further). Also, knowledge changes over time and continuous maintenance is often required. This creates the need for editing tools that enable a domain specialist to update the knowledge in the system.
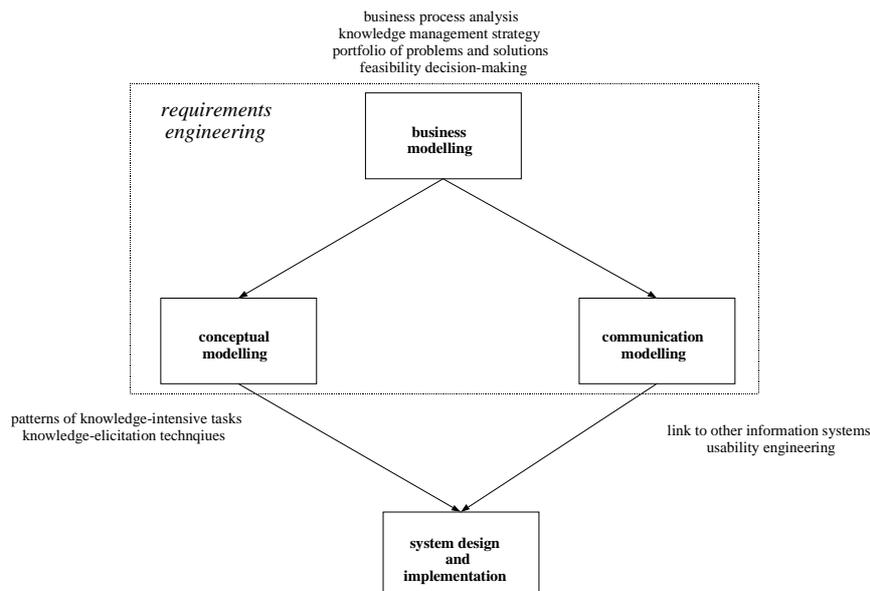
Figure 3: Activities in KBS construction

Figure 3 gives an overview of the major activities in KBS construction. Please note that the upper part of this figure can be seen as requirements-engineering activities: conceptual modelling provides the reasoning requirements, communication modelling the interaction requirements, and business modelling the other external and user requirements.

1.5   Link to O-O modelling

A KBS cannot be constructed in isolation. A KBS typically functions as a knowledge-intensive component in an IT environment. It is therefore important that clear links exist between KBS development and other development activities. Over the past years the KBS methodologies have started to make this link by adopting as much as possible techniques and notations from the object-oriented community, in particular the Unified Modelling Language (UML, Booch et al. 1998). For example, the CommonKADS methodology uses many UML notations. To make this link even more explicit, Van Joolingen et al. (1999) have constructed a handbook for KBS construction in an object-oriented fashion.

Integration of a KBS also becomes more feasible with the advance of component-based development frameworks. In such an approach a KBS can be integrated as a knowledge-intensive component within a larger system.

1.6   Link to knowledge management

In recent years, knowledge management has become a focus of attention for many organizations. Knowledge is considered to be the key source for sustainable competitive advantage (Nonaka & Takeuchi 1995; Davenport and Prusak 1998). Therefore, Knowledge management is aimed at locating, capturing, transfering, sharing and creating knowledge within and across organizations (Von Krogh, et al. 1997; Von Krogh et al. 2000). It will be clear that conceptual modelling, as developed within the field of KBS construction, provide key techniques for knowledge management (Gaines, Musen & Uthurusamy 1997; Shadbolt & Milton 1999; Wielinga, Sandberg & Schreiber 1997; Wigg, De Hoog & Van der Spek 1997; Schreiber et al. 2000).

Knowledge mapping is a good example of a useful knowledge management activity with existing conceptual modelling techniques at its foundations (Vail III 1999; Speel et al. 1999). Knowledge mapping creates high-level knowledge models in a transparent graphical form. Using knowledge maps, management can get an overview of available and missing knowledge in core business areas and take appropriate knowledge management decisions.

2   BUSINESS MODELLING

Business modelling is concerned with modelling business processes in which we are interested from a knowledge point of view. The reason for creating business models is that they provide a description of the overall context in which the knowledge model must function. It allows for an analysis of the *actual* needs for knowledge-intensive applications, and gives a first inventory of the knowledge that needs to be modelled. In this section, we briefly describe the components of a business model. The techniques that are needed to fill the model are not discussed here. Usually these include interviews with persons at various positions in the organization, as well as an analysis of the organization's documentation.

2.1 Bsiness model

Business models describe (a larger part of) an organization (Checkland and Scholes 1990; Scott Morgan 1994; Johansson et al. 1993, Watson 1994, Eriksson and Penket 1998). The focus is not (yet) on one particular software system. The description of a business model typically includes (see Akkermans et al., 1999 and Schreiber et al., 2000 for details):

- Problems and opportunities in a wider organizational context

- Organizational context - key features of the wider organization to put the problems and opportunities into perspective (Mission, vision, strategy, objectives, external factors, value chain and value drivers)

- Potential solutions to the problems and opportunities

- System context model - variant aspects of the organization that may change as a result of the introduction of a knowledge based system:

- Structure - organization chart of the considered (part of) the organization in terms of departments, groups, units, etc.

- Business processes - layout of the relevant business processes (for example by a flow diagram)

- People involved, including decision makers, knowledge providers and knowledge users

- Resources that are utilised for the business processes, including information systems, equipment, materials, patents, technology, rights

- Knowledge, a special resource expoited in the business processes (to be described on a high level)

- Culture & power, 'the way we do things around here' including networks, communication, knowing who knows what, social issues, politics, serendipity and trust

- Process breakdown - the business processes broken down in smaller tasks. Namely, the KBS should be focused on a specific task. Tasks descriptions include people who carry out the task, business relevance (in terms of frequency, cost, competitive advantage), knowledge intensity, and knowledge assets. The business process is broken down to a level of detail where tasks can be identified where a KBS can bring added value.

- Knowledge assets that need to be exploited in the business processes. A knowledge asset description should include its custodian, business processes it is used for, whether the knowledge asset has the right content, is represented in the right form, is at the right place at the right time and whether the knowledge asset is managed (creating new knowledge in the area, maintaining, disseminating and exploiting the knowledge).

Each application project should specify a general business model, relating to the relevant part of the organization for the project. Ideally, this model is already present for a large part. The business model is usually descriptive, using plain text, organisation diagrams as well as notations for general business processes.

2.2 System Context Model

The second part of a business model is also called a system context models. A system context model describes the direct organizational environment of a software system. System context models are typically used in application development projects in order to indicate how the system should interact with its environment.

System context models typically have a smaller scope than business models. Their content is more system-analysis oriented. System context models describe information and control flow between the system and its environment.

A system context model is required for every application. Note that this model is sometimes called a "business model" as well. Its scope is, however, limited to the direct environment of the system, in other words, the system context model describes the direct environment of a (knowledge intensive) system and the interaction of the system with this environment.

The technique of choice for specifying a system context model is the UML use case diagram (Jacobson et al. 1999). An extended notation for this type of diagram has been introduced by the Catalysis method (D'Souza & Wills, 1998). This notation appears to be very useful to model the system context, since it can be both very general and precise at the same time.

Furthermore, the system context model can include, when relevant:

- A UML state diagram or a UML activity diagram of the interaction control between the external actors and the system, in other words a diagram describing the timing of the interaction between the system and its environment.

- An activity diagram can display major object flows, indicating the flow of information in the organization.

- A UML class diagram, with all classes representing types, to describe the static structure of the information exchanged between the external actors and the system.

- A description of the knowledge assets that the system should own.

- A description of all the external actors in the use case diagram.

There should be two instances of the business model: one analysing the current organization, and a second one designing a future, desired organization. When a KBS solution is chosen, this KBS forms part of the future business model. Critical for a successful deployment of a KBS solution in the organization is the application of change management techniques.

2.3   An Example: Assessing Mortgage Applications

To illustrate business modelling, consider a bank providing mortgages. The director, responsible for mortages in the company, is checking the statistics with respect to mortgages over the last 2 years. After a detailed study and some additional fact finding, he finds out that at certain bank locations problems occurred with clients who could not pay off the mortgage debts. He concludes that some sales managers take big risks. After some thinking, he also studies the facts of other bank locations. Although this problem did not occur, these banks have a much lower turnover. After some additional fact finding, he discovers that the financial specialists are relatively older. He hypothesizes that these specialists might be more conservative and therefore might misjudge adequate mortgage applications as too risky. As a conclusion he decides that the decision criteria to grant a mortgage should be aligned. This will result in a higher turnover under restricted financial risks.

The director initiates a project and explains the leading knowledge manager his observations and direction. The knowledge manager interviews some additional directors, sales managers, financial specialists and clerks to understand that organization context. From the general director, he understands that a new vision and strategy has just been formulated. The bank misses a lot of turnover since the regular clients becoming older and older. They will now target a much younger group of people. This might give more risks, however, the increase in turnover will compisate this. This will result in a strong growth again.
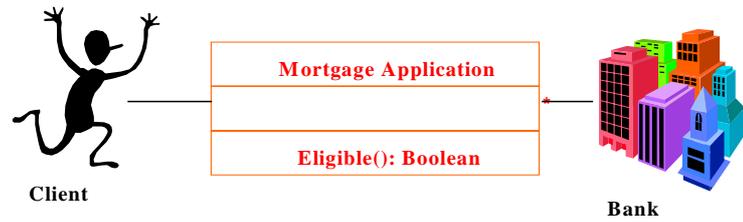
Figure 4: System context diagram for a mortgage application

The sales managers and financial specialists which are interviewed by the knowledge manager indeed differ enourmously. Some young sales managers do this job just for a short period and try to reach a short-term growth which is important for their further career. On the other hand, more senior financial specialists think in decades while assessing mortgage applications. This results in more conservative judgement and especially young couples with changing jobs are considered too unstable for a long-term mortgage.

Finally, the knowledge manager interviews some clerks who don't have any responsibility with respect to mortgage applications. These sessions turn out to be very useful since clerks are the first point of contact for clients. The knowledge manager discovers that clerks advise clients on their personal judgement whether there is a chance for success in specific mortgage applications.
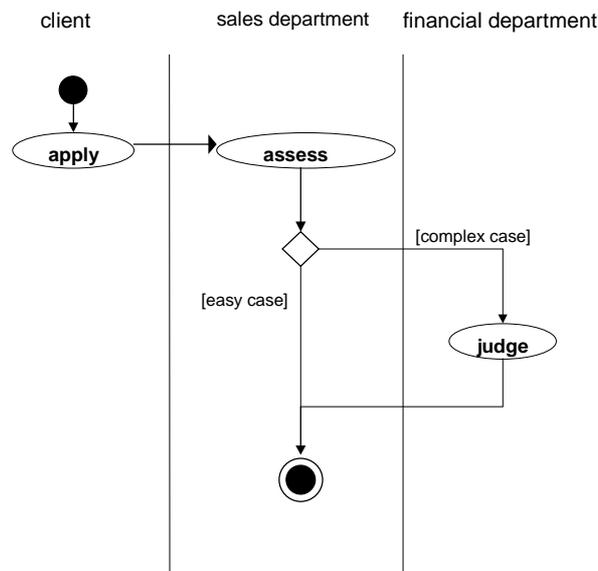

Figure 5: Activity diagram for the application and acceptance process of a mortgage. The mortgage is applied by the client and first assessed by a sales manager. In easy cases he can handle the application himself, otherwise a specialist from the financial department must judge the application based on financial expertise.

Based on her findings, the knowledge manager concludes that sales managers, financial specialists as well as clerks should align in decision criteria to grant a mortgage. She suggests two potential solutions:

1. One way of doing this is organising a training course for all of them. However, it might be difficult to find an adequate teacher who knows the adequate decision criteria and is able to train both conservative senior financial specialists as well as young career-minded sales managers. In addition, the training should be repeated frequently due to the high job rotation rate of sales managers.

2. A second option is to make the knowledge on assessment of mortgage applications explicit. This can be done in knowledge acquisition sessions with both sales managers and financial specialists. The resulting business rules can then be presented to the director who can confirm or adjust them based on the bank's vision. These business rules form then the basis of a KBS which will advise the sales manager and financial specialist.

The process of applying for a mortgage and assessing it is displayed in Figures 4, 5 and 6. A client applies for the mortgage, the bank will assess the application and the client will be informed whether the loan is granted or not. The mortgage application forms the interface between the client (filling in the application form, answering questions) and the bank (processing, assessing and finalizing the application). The post condition indicates that the end result of the action is that the client is notified of the result, and that the result is a consequent of applying the business rules (financial as well as legal and policy knowledge) that the bank uses in assessing this kind of applications. The "assess" task will be marked as *knowledge intensive,* meaning that business-critical knowledge (the business rules) is required to perform the task. In a glossary, all terms (processes, types, attributes, conditions, etc.) needs to be defined. For lack of space, further details of the system context model are not depicted here.

After spending about 2 months interviewing people, collecting information, filling template models, a SWOT analysis and identifying risks, the feasibility study is completed. In a presentation to the board of the bank, the knowledge manager recommends to develop a KBS together with a change programme to deploy the system. The board agrees on the recommendation and decides that all clerks, sales managers as well as financial specialists will be trained as part of the change programme. This way, more applications can be handled by the sales manager without taking high risks. In addition, the board makes the use of the KBS compulsary for each mortgage application. However, since people have final saying, the recommendation of the KBS may be overruled. However, a written explanation needs to produced when this happens. These explanations then form the basis for updates of the KBS.
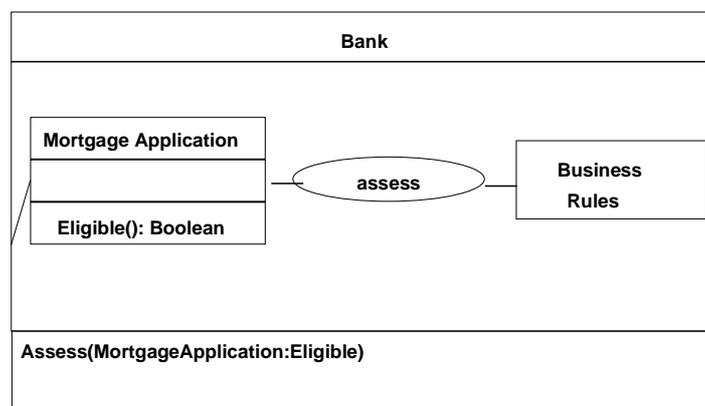

Figure 6: Assessment of a mortgage application. Input is the application as well as business rules (the norms). Output is the 'eligible' decision.

3 CONCEPTUAL MODELLING

Detailed requirements engineering is split into two parts in the CommonKADS approach. The knowledge model specifies the knowledge and reasoning requirements of the prospective system; the communication model specifies the needs and desires with respect to the interfaces with other agents: i.e., a user interface and/or interfaces with other software systems. Together, the knowledge model and the communication model form the input for system design and implementation. Input for the conceptual-modelling process is a certain task identified during business modelling. We assume that the task selected is characterized as being knowledge intensive, and that conceptual modelling of the task and its related knowledge is considered feasible from a technical, economical as well as a project perspective.

Conceptual modelling[1] is a technique that helps to clarify the structure of a knowledge-intensive business task. The knowledge model of an application provides a specification of the data and knowledge structures required for the application. The model is developed as part of the analysis process. It is therefore phrased in the vocabulary of the application, meaning both the domain and the reasoning task:

- *Domain Knowledge* specifies the domain-specific knowledge and information types (e.g., houses, finance, equipment, materials) that we want to talk about in an application. A domain-knowledge description is somewhat comparable to a data model or class model in software engineering.

- *Task/Inference Knowledge* describes what goal(s) an application pursues (e.g., assessment, diagnosis, configuration), and how these goals can be realised through a decomposition into subtasks and (ultimately) inferences. This "how" aspect includes a description of the dynamic behaviour of tasks, i.e., their internal control. Task knowledge is similar to the higher levels of functional decomposition in software engineering, but also includes control over the functions involved. Inference knowledge describes the basic inference steps that we want to make using the domain knowledge. Inferences are best seen as the building blocks of the reasoning machine. In software engineering terms the inferences represent the lowest level of functional decomposition.

The knowledge model does *not* contain any implementation-specific terms. These are left for the design and implementation phase. It is essential that during conceptual modelling, implementation-specific considerations are left out as much as possible. For example, when we talk during analysis about "rules," we mean the rules human specialists talk about. Whether these natural rules are actually represented in the final system through a "rule" formalism, is purely a design issue, and not considered relevant during analysis. This clear separation frees the analyst from worries concerning implementation-specific decisions. This requires that the analyst has experience in developing knowledge models which are "designable."

Note that the knowledge model has a structure that is in essence similar to traditional analysis models in software engineering. The reasoning task is described through a hierarchical decomposition of functions or "processes." The data and knowledge types that the functions operate on are described through a schema that resembles a data model or class model.

---

[1] In the current section we will describe the approach to conceptual modelling prescribed by the System Development Framework, version II (Van Joolingen, Schreiber, & Hermans, 1999). SDF-II combines the conceptual modelling principles of CommonKADS (Schreiber et al, 2000) with object oriented modelling methods and –notations. As notation language it uses the Unified Modelling Language (UML, Booch et al, 1999). SDF-II is presented as an advanced method for modelling knowledge in an object oriented way. It combines the long experience in knowledge modelling laid down in the CommonKADS framework with the latest insights in OO-modelling.

3.1 Conceptual Modelling of Domain Knowledge

Modelling domain knowledge implies capturing the static structure of information and knowledge types. Just like in regular data modelling, a schema is constructed containing the major types and relations occurring in the application domain. For example, in the mortgage application we have information types for both the client and the mortgage, as well as a "mortgage application" relationship between them. Part of such a schema is shown in Figure 7. The notation used is similar to a UML class model. In practice, we would need to identify the various attributes needed to describe a client, mortgage as well as a mortgage application.
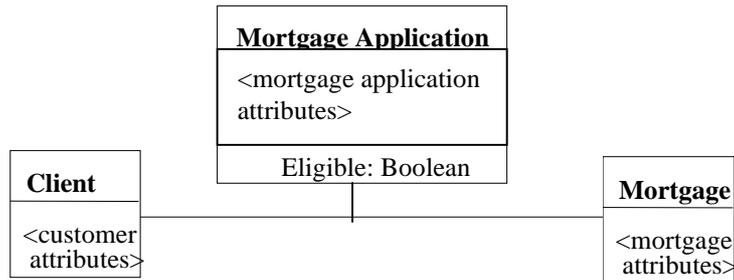

Figure 7: Information types in the mortgage application. The Application relation between a Clinet and a Mortgage is modelled through a UML association class construct.

In the knowledge engineering community the term *ontology* is used for schemas which capture the understanding of the domain specific vocabulary (Gaines, 1997). An ontology is defined as an explicit specification of the terms and their meaning in the domain of interest. For example, a medical ontology could specify that in the medical domain one can distinguish diseases, syndromes, symptoms, therapies etc. and diseases may cause symptoms, but not vice versa. Currently, the knowledge engineering community is investigating to what extent and under which conditions ontology specifications can be 'decontextualised' and indexed, so that reusable ontology components can be offered to knowledge engineers.

In addition to schemas or ontologies, we need to define a number of different knowledge bases. In the early days of expert systems a single large knowledge base was constructed. In practice, this is cumbersome, because such a knowledge base is difficult, if not impossible, to maintain. Therefore, we try to distinguish a number of different knowledge bases containing sets of rules with a similar purpose and/or structure. We do not have a generic approach how to best classify rules in knowledge bases. This is part of the experience of a knowledge engineer. One heuristic is to organise rule sets according to reasoning steps (see the next section). However, rules might be applicable to several reasoning tasks.

Figure 8 shows three sample knowledge bases in the mortgage application domain with some sample rules. Techniques exist for modelling the structure of rules (Schreiber et al., 2000, Chapter 5). Such a technique is useful, because knowledge bases need to be frequently updated and maintained. The notion of a "rule type" can significantly enhance this maintenance process and can be used as a basis for rule elicitation interfaces that interact with a domain specialist. Within rules as well as knowledge bases, domain knowledge needs to be clearly separated from task knowledge.
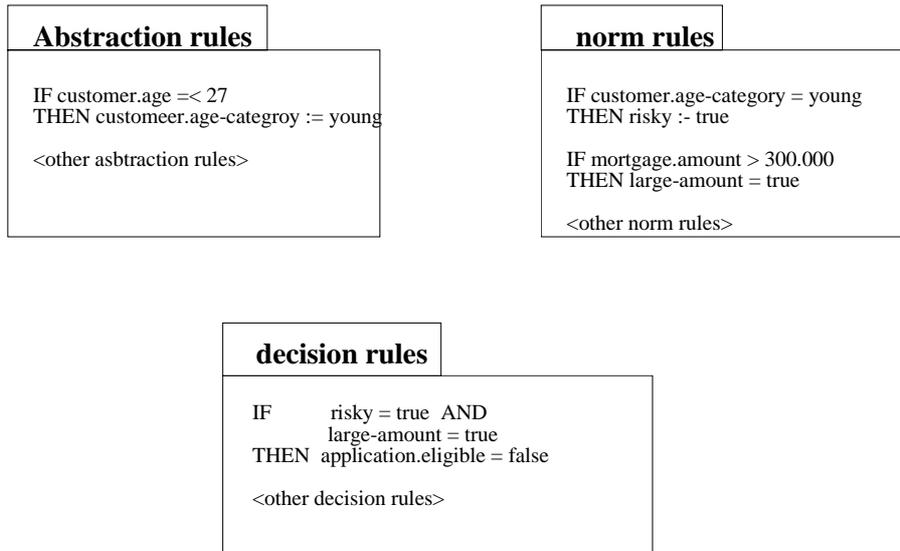
```
┌─────────────────────────┐        ┌─────────────────────────────────┐
│  Abstraction rules      │        │  norm rules                     │
├─────────────────────────┴──┐     ├─────────────────────────────────┴──┐
│                            │     │                                    │
│  IF customer.age =< 27     │     │  IF customer.age-category = young  │
│  THEN customeer.age-categroy := young │ THEN risky :- true            │
│                            │     │                                    │
│  <other asbtraction rules> │     │  IF mortgage.amount > 300.000      │
│                            │     │  THEN large-amount = true          │
│                            │     │                                    │
│                            │     │  <other norm rules>                │
└────────────────────────────┘     └────────────────────────────────────┘
```

```
        ┌─────────────────────────┐
        │  decision rules         │
        ├─────────────────────────┴──┐
        │                            │
        │  IF      risky = true  AND │
        │          large-amount = true │
        │  THEN  application.eligible = false │
        │                            │
        │  <other decision rules>    │
        │                            │
        └────────────────────────────┘
```

Figure 8: Three knowledge bases containing three rules sets for solving the assessment problem

3.2   Conceptual Modelling of Task/Inference Knowledge

Task/inference knowledge describes the objectives of an application together with a method *how* to achieve these objectives. *Tasks* form the top-level unit of analysis for the task layer. A task can be, for instance, assessing a mortgage application, diagnosing a patient, or making a lesson schedule for a school semester. Tasks can be decomposed into subtasks or into basic inferences. Upon decomposition, the control over the subtasks should be specified: which tasks to perform first, tasks to iterate, etc.

Tasks are divided into subtasks up to a level of elementary *inferences* that are not decomposed further. As a result, a task is composed of a number of combined inferences yielding an *inference diagram*. An inference specifies a basic step in a reasoning process in terms of inputs, outputs and the knowledge needed for the step, as depicted in Figure 9. Knowledge roles establish the link between task/inference knowledge and domain knowledge.
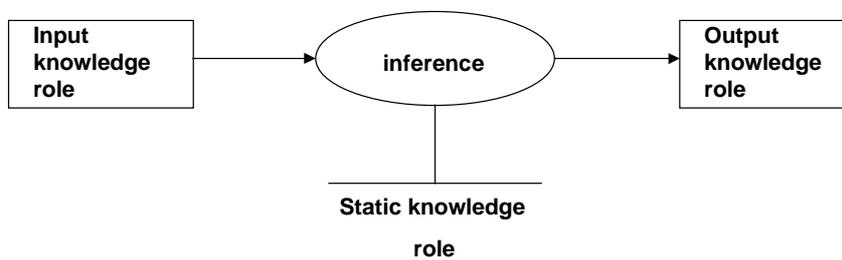

Figure 9: A CommonKADS inference (using CommonKADS notation)

We distinguish two types of knowledge roles, namely dynamic knowledge roles and static knowledge roles. *Dynamic knowledge roles* are the inputs and outputs of inferences. *Static knowledge roles* are more or less stable over time. Static knowledge roles specify (reformulations of) collections of domain knowledge that is used to make the inference.

Examples are the set of rules that are used to assess a person's file, heuristics to find possible solutions of a problem, and taxonomies to classify cases.

The coupling between inferences and domain knowledge enables us to reuse inference descriptions independently from domain knowledge. Libraries of reusable task/inference components are available in the form of problem-solving methods (Benjamins 1993, Breuker and Van de Velde 1994, Puppe 1993, Benjamins et al. 1996, Motta 1999). Also domain knowledge (ontologies as well as knowledge bases) can be reused independently from inferences.

3.3   Assessing a mortgage application

As described in the previous section, the board of the bank decided for the development of a KBS to assess mortgage applications. The knowledge manager starts the next phase in the project which is conceptual modelling for KBS development. In order to prevent 'reinventing the wheel' let us see whether generic assessment components can be reused. In Figure 10, the generic assessment task is presented which can easily be mapped on the mortgage application assessment task in Figure 6. In order to discuss the mortgage assessment application in more detail, we first focus on reusable structures for the assessment task.
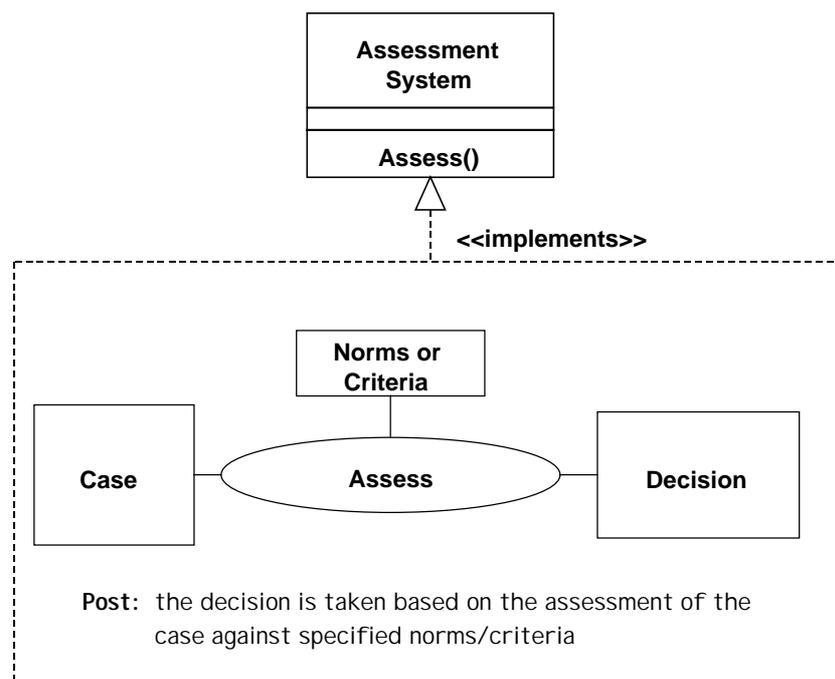

Figure 10: The assessment action implementing the assessment responsibility of the assessment system, explicitly making use of knowledge.

The goal of an assessment task is to evaluate a particular case against some norms or criteria and take a decision. In the example, a decision needs to be made whether to accept a mortgage application. The input knowledge role consists of data about the case, i.e., the mortgage application. The static knowledge role consist of norms or criteria which are used to evaluate the case in order to take a decision. These criteria include for example business rules that relate income to the amount requested. The ouput knowledge rule consist of the decision, eligible for a mortgage, yes or no.

For the decomposition of the assess task we made use of the assessment template model from CommonKADS. See Figure 11 for this decomposition which is widely applicable to simple assessment tasks. The method consist of the following inferences:

- **Abstract case.** Most of the time some of the case data need to be abstracted. For example, the age of the applicant need to be abstracted. The abstractions required are determined by the data used in the norms. The abstracted features are added to the case.

- **Specify norms.** The norms or criteria that can be used for this case need to be find. In most assessment tasks, the norms are dependent on the case. An example of a norm would be 'loan amount matches income'

- **Select norm.** From the set of norms generated by the previous inference, one norm needs to be selected for evaluation.

- **Evaluate norm.** Evaluate the selected norm with respect to the case data. This function produces a truth value for the norm. For example, 'loan amount matches income is false'. This function is usually a straightforward computation.

- **Match.** This inference checks whether the results of the evaluation leads to a decision. Sometimes, the truth value of one norm is sufficient for a decision. For example, for the 'loan amount matches income', the decision found by the match function is 'not eligible for this mortgage'
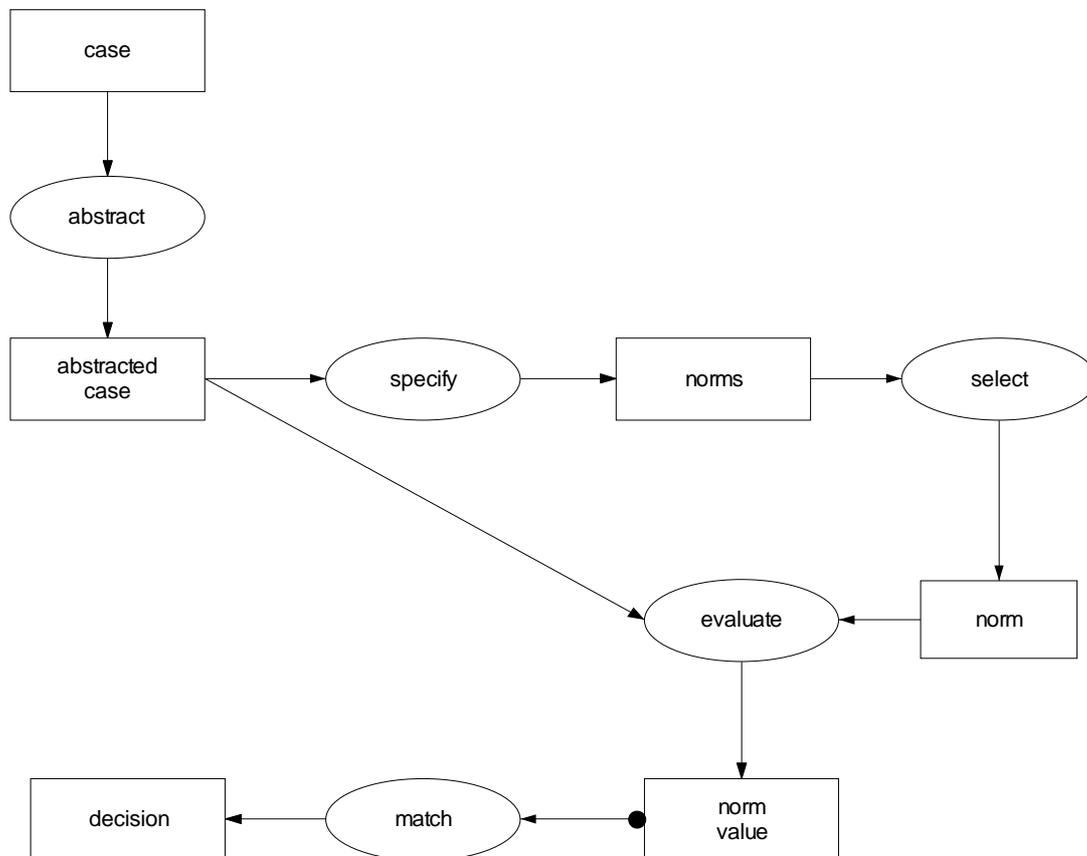
Figure 11: Generic inference diagram for an assessment task

14

An important issue in interpreting diagrams like Figure 11 is that of *control.* Please note that the diagram does not specify the order in which the actions take place. *Activity diagrams* can be used to define a full control structure. Figure 12 defines a specific control structure on the actions defined in Figure 11. The activity diagram shows a structure for data driven inference and a control structure over all actions present in the diagram, including iteration.
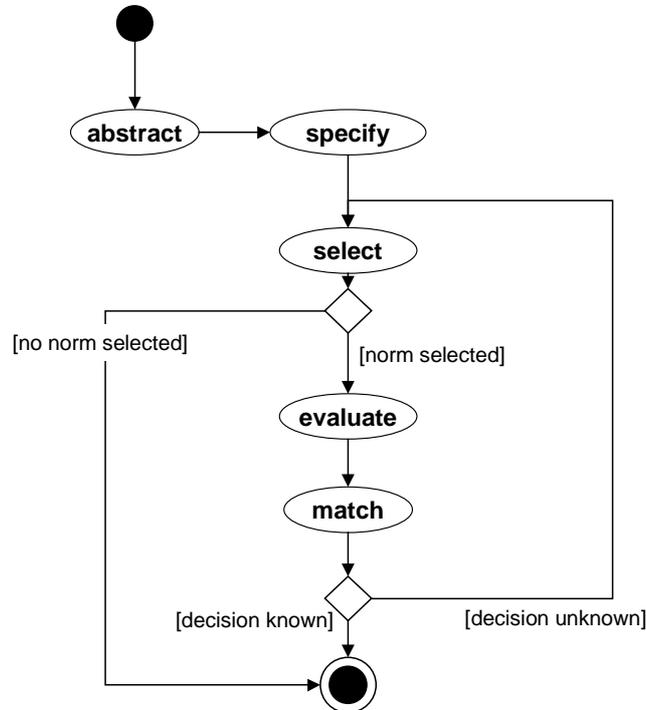

Figure 12: Activity diagram defining a control structure for data driven inferences using the actions in Figure 11

Along with the decomposition of the assessment task, also the domain knowledge needs to be aligned. The object of type 'Business Rules' depicted in Figure 6 represents the whole of knowledge that is needed in the system to complete the assessment operation. When this operation is decomposed, the knowledge should also be decomposed in order to indicate which parts of the knowledge are needed for which parts of the action. We can distinguish the following main information and knowledge types.

- Case data. This might be attributes of domain type 'Mortgage Application' which includes links to 'Client' and 'Mortgage' (Figure 7).

- Case abstraction knowledge (Figure 8).

- Norms or criteria. Business rules on attributes of domain types against which the case can be accessed (Figure 8).

- Norm-evaluation knowledge. Specifications of logical dependencies between case data and norms or criteria. Norm ordering knowledge can be added as well.

- Decision knowledge. Specifications of the decision options that can act as task output, as well as logical dependencies between norm values and a particular decision (Figure 8).

3.4 Guidelines for Conceptual Modelling

In the previous sections, we have mainly concentrated on the content of the knowledge model. The next question is how to undertake the process of conceptual modelling. This is a difficult area for which no single solution exists. We will present some general guidelines that have proved to have worked well in practice. For more details, we refer to Schreiber et al. (2000).

We distinguish three stages in the process of conceptual modelling.

Knowledge Identification

Information sources that are useful for conceptual modelling are identified. This is really a preparation phase for the actual knowledge model specification. A glossary of domain terms is constructed. Existing model components such as task templates and domain schemas are surveyed, and components that could be reused are made available to the project. Typically, the business model forms the starting point for this stage.

Knowledge Specification

In this second stage the knowledge engineer constructs a specification of the knowledge model. First, a task template is chosen and an initial domain schema is constructed, using the list of reusable components identified in the previous stage. Then, the knowledge engineer will have to fill in the holes in the knowledge model. There are two approaches to complete the knowledge model specification, namely starting with the inference knowledge and then moving to related domain and task knowledge, or starting with domain and task knowledge and linking these through inferences. The choice of approach depends on the quality and detailedness of the chosen task template (if any). In terms of the domain knowledge, the emphasis in this stage is on the domain schema, and not on the knowledge instances that belong to the knowledge bases. This can be left to the next stage.

Knowledge Refinement

In the final stage, attempts are made to validate the knowledge model as much as possible and to complete the knowledge bases by inserting a more or less complete set of knowledge instances. An important technique for validating the initial specification that comes out of the previous stage is to construct a simulation of the scenarios gathered during knowledge identification. Such a simulation can either be paper based or involve the construction of a small, dedicated prototype. The results of the simulation should give an indication whether the knowledge model can generate the problem-solving behaviour required. Only if validation delivers positive results it is useful to spend time on completing the knowledge bases.

These three stages can be intertwined. Sometimes, feedback loops are required.

An important technique to create knowledge models is called *knowledge acquisition* which is described in the following section.

4 KNOWLEDGE ACQUISITION

The previous section emphasized the importance of the knowledge model as an intermediate representation of the knowledge that bridges the gap between knowledge as it is viewed by people in the field (often also the users of the KBS) and the knowledge as it is represented in the system. The knowledge model has sufficient authenticity to be meaningful to the domain knowledge specialists, as well as sufficient structure to allow a system developer to further develop the KBS without assistance of a domain specialist.

4.1 Knowledge Acquisition Techniques

The question then arises how such a model is to be build from all the experiences the specialists have. This field of elicitating knowledge and capturing it in a structured format is called *knowledge acquisition* (McGraw & Harbison-Briggs 1989; Shadbolt & Burton 1990; McGraw & Harbison 1997; Scott, Clayton & Gibson; Shadbolt, O'Hara, & Crow, forthcoming). In a knowledge acquisition dialogue, the knowledge engineer helps the domain specialist to make his/her implicit (or tacit) knowledge explicit. For this, numerous techniques have been developed (see Table 2).

Technique	Description
Repertory grid	The repertory grid technique consists of two phases. In the first phase, domain constructs are collected by means of triadic elicitation. Three domain concepts are shown to the specialist who is invited to say which of the three concepts are more similar and which one is different. By repeating this a number of times, the dimensions that specialists use are explicated to organise knowledge in the domain of reference. In the next phase, all domain concepts are scored on the dimensions elicited in the first phase, this way creating a complete picture of the relevant concepts and their properties. Repertory grids are mainly used in the context of eliciting knowledge for classification tasks.
Laddering	Laddering was primarily developed as a technique for eliciting concept hierarchies. By means of a structured interview the domain specialist is invited to put all the concepts relevant to a knowledge domain in a type/subtype hierarchy. However, laddering has also been reported useful for eliciting other types of relations between concepts (e.g. causes, part-of, etc.). Laddering can be used to elicit both concepts and relations.
Card sorting	Card sorting is a very general technique to classify domain concepts. Two forms can be distinguished: free format sorting and guided sorting. With *free format sorting*, the specialist is given cards with concepts and asked to make piles of concepts in any way he considers meaningful. This way, the technique mainly supports the elicitation of dimensions in a way similar to the first phase of the repertory grid technique. With *guided sorting*, the piles are fixed, and the specialist is asked to assign the cards to these piles. This is similar to the second phase of the repertory grid technique.
20 questions	The 20 questions technique is basically a game where one specialist thinks of a task or concept while an other specialist tries to find out what that task is by asking maximally 20 questions. The nature of the questions and the order in which they are asked then form the basis of an interview where the knowledge engineer tries to explicate the strategy used by the specialist. This technique is mainly useful in the context of identifying the reasoning method (inference structure in CommonKADS) used by the specialist.
Protocol analysis	In protocol analysis (also called think aloud method), the specialist is given a description of a problem and is asked to verbalise all his/her thoughts while solving the problem (e.g. making a diagnosis). The protocol is then later analysed by the knowledge engineer to identify the structure of the reasoning process. Like the 20 questions technique, this technique is mainly useful in the context of identifying the reasoning method.

Table 2: Knowledge acquisition techniques for making implicit knowledge explicit

Tools for knowledge acquisition techniques are typically targetted at particular techniques and particular types of knowledge. Experimental evaluation has shown that a combination of different techniques can uncover a range of types of knowledge in a particular application (Shadbolt et al. forthcoming). This leads onto the idea of presenting knowledge acquisition techniques as a connected series of tools in a workbench (Anjewierden et al. 1990). PC PACK, for example, is a PC-based portable package of tools for knowledge acquisition and engineering, and requirements capture (O'Hara et al. 1998). More than a dozen tools are

integrated in this workbench, together with a common underlying knowledge representation language.

4.2   Knowledge Acquisition Bottleneck

Although methodologies such as CommonKADS support the knowledge acquisition process in a number of ways (e.g. by providing modelling constructs and template models) experience shows that conceptual modelling remains a difficult and time-consuming activity. As reported by Musen (1994), at least four reasons can be identified for the so-called knowledge acquisition bottleneck: (i) the problem of tacit knowledge, (ii) the problem of communication (iii) the use of representations, and (iv) the fact that modelling is a creative task.

Tacit knowledge

Since the knowledge model is intended as bridge between domain specialists and knowledge engineers, the model can only be developed in a dialogue between the two. The domain specialist is needed to provide the content of the knowledge while the knowledge engineer is needed to ensure that the available constructs in the modelling are used properly. Unfortunately, domain specialists often have difficulties in verbalising the knowledge they use to solve problems. The reason for this is that the knowledge they use is often only available in a tacit, non-verbalisable form. A distinction can be made between at least three forms of knowledge:

- Explicit knowledge: knowledge that is persistently represented in a disembodied form (e.g. on paper, microfiche, knowledge model, …)

- Implicit knowledge: knowledge that is not explicit but that easily could be made explicit (e.g. by writing it down)

- Tacit knowledge: knowledge that is not explicit, and that also cannot be made explicit easily.

Ideally, the knowledge engineer helps the domain specialist to make his/her implicit knowledge explicit in a knowledge acquisition dialogue. Unfortunately, specialist reasoning is often based on partially tacit knowledge. The specialist knows how to solve the problem, but is unable to explain this. This knowledge is often purely experience based, and often crucial for specialist problem solving (research in psychology has demonstrated over and over that it is typically the tacit, experience based knowledge that distinguishes specialists from novices).

This is not to say that it is impossible in principle to make tacit knowledge explicit, but it is difficult and the standard knowledge engineering repertoire does not include techniques to support this.

Miscommunication

The second cause of the knowledge bottleneck identified by Musen is that of miscommunication. Although domain specialists and knowledge engineers usually are able to talk to one another, true understanding is often hindered by the fact that they use similar words but mean different things. To take an obvious example: for medical specialists and computer specialist the term virus has a completely different meaning. In this case, both meanings of the term are widely known, so it won't lead to miscommunication. However, in more subtle occasions of this phenomenon the difference in meaning will often be unnoticed, leading to sometimes absurd dialogues between specialist and knowledge engineer.

To resolve this problem, knowledge engineers should during the knowledge identification phase reserve time to develop an understanding of the domain specific vocabulary. One way to do this is to develop an explicit domain ontology (Gaines, 1997).

Knowledge representations

Knowledge models are usually expressed in a (pseudo) formal language. Formal languages have the virtue that they allow some things to be expressed more easily than others, while other things may not be expressible at all. In many occasions this is indeed a virtue, because it forces the knowledge engineer and the domain specialist to express the knowledge in a canonical, analysable form. However, sometimes it hinders the knowledge modelling process because the formalism does not allow the knowledge to be expressed in a natural way. Although this problem was more severe in the days when knowledge engineers tried to express the knowledge directly in an executable form (e.g. production rules), our experience is that it still exist in a lesser form with current knowledge modelling languages, including the CommonKADS modelling language and its UML counterpart.

Modelling is creative

Knowledge models are not simply the result of making an implicit model explicit. Instead they are the result of a creative, often trial and error based endeavor and often the domain specialists are just as surprised by the result as the knowledge engineer.

5   TOOLS AND TECHNIQUES FOR PROTOTYPING

KBSs are a type of software solution in which automatic problem solving takes place, as opposed to conventional software systems which are primarily concerned with the treatment of data (input, output, printing, aggregation). For the kinds of tasks that can be supported (or sometimes even automated) with this technology: see Figure 2. In KBSs "frozen" or static knowledge is implemented in a software program, for example, in the form of rules or decision trees.

In this section we restrict ourselves to tools and techniques for building KBSs. Besides these tools there are tools for datamining (discovering unexpected relations c.q. knowledge in data), tools for simulation, tools for perception, robotics, language processing, etc. which are all not considered in this article. Note that tools for machine learning ("fluid" knowledge) applications are left aside as well.

In Figure 13, a general architecture is given for KBSs. It is a layered architecture that can be read as follows: an IT solution is made of hardware, an operating system, middleware and a programmed software application. This software application consists of different layers (could be classes, for example). The most important layer in KBSs is the knowledge layer. This layer can be subdivided into three layers representing task, inference and domain knowledge as mentioned in the section regarding conceptual modelling.
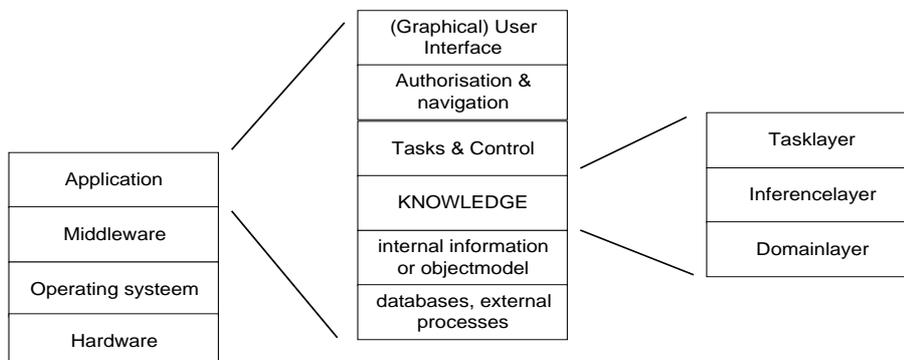

Figure 13: A general architecture is given for KBSs

Any programming language for building KBSs should be able to provide means to program each layer or at least have access to elements in other layers in a standardised, open way.

In the next sections we will discuss in general the benefits of KBS development environments, some historical considerations, some details of KBS programming languages and finally an outlook to the future.

5.1 Tools for building knowledge based systems

Two main benefits of using special tools for building KBSs are lower costs for development and maintenance and a shorter time-to-market compared to building the same things using general programming tools. The reason why these specialised tools can offer such advantages are their built-in knowledge representation and manipulation methods plus the integration of these methods with other parts of a program like the object model and the user interface elements. It gives a programmer the possibilities to focus on the essence of the job: implementing knowledge models.

One could use general purpose programming languages like C++ or Java for building KBSs. Sometimes this is done, for instance to achieve the smallest memory footprint or the ultimate performance. On the other hand, this has some disadvantages.

For each knowledge representation a corresponding algorithm is computationally necessary to process the knowledge items. One has to encode the algorithms and representations of rules oneself in the language one is working in. You'll redo work that has already been done. This probably will cost a lot of (extra) time and effort and could also have a negative impact on the quality of the final software product that is developed. It will be hard to achieve the run-time performance current tools have. Years of investment have been put in these tools. An alternative is to buy a library or component that contains the required functionality. Nowadays professional libraries can be found on the market.

In the early days of KBSs, languages like LISP and Prolog were used which were popular in the Artificial Intelligence community. For both languages different development environments were and are built by different software vendors and are and have been used for different kinds of applications.

Since then, the rule formalism has been very popular for KBS development. In the '80s new tools entered the market, focussing on facilities for programming knowledge representations like rules and decision trees. Some tools came in the form of components, for instance Haley's RETE++, a rule based library to be used together with C++, or ILOG's planning & scheduling components. Examples of complete application development environments that ranks among the leaders in the market nowadays are Computer Associates' Aion, marketleader in financial sector and also Gensym's G2, marketleader in the industrial sector.

Currently, many types of knowledge representation can be used and processed in computer programs: object-orientation, decision trees, (hyper) graphs, blackboards, (many different kinds of) logic, cases (like in jurisdiction) and many more.

For each knowledge representation you can find one or more languages on the market. For each language, you can find one or more programming or development environments and/or components on the market. There are a number of websites that give you an overview of the current available tools and languages, for instance

> http://aiintelligence.com/aii-info/aiibank.htm

These and other tools have many characteristics. This short section does not leave much space to go into detail. One can distinguish the following main features:

1. Knowledge representation facilities: one should be able to express oneself as elegant as possible (implementation rules should resemble business rules as much as possible) and with as little characters or symbols as needed
2. Development and / or maintenance features and, thus development time, costs or quality
3. General information regarding the supplier
4. Developer's support and user base
5. Integrated development environment features
6. Target environment services (operating systems, database, middleware etcetera)

5.2 Continuous developments & refinements

Throughout the years lots of things have changed in the area of information technology. The market regarding KBSs is following these developments:

- General programming languages changed character from symbolic to structured to procedural to object oriented and recently into component based development languages.

- The hardware and infrastructure changed from mainly mainframe based to micro computers to personal computers to client-server computing and finally into web-based computing, all having their own new or revised characteristics and standards.

- Continuous improvements of general programming facilities, like graphical user interface construction, syntax highlighting, database connectivity, incremental testing and debugging, multiplatform support, etc.

- The use of and connection to different standards changes rapidly throughout the years: RPC, SQL, ODBC, JDBC, ORB, HTML, XML, etc.

- The way of working changed from primarily a waterfall methodology to an incremental, iterative way of working, in which reuse of components is a built-in feature.

Design tools that are currently used in the area of knowledge based systems are object oriented case tools like Rational's Rational Rose and Computer Associates' Paradigm Plus. These tools do not contain specific knowledge-based elements and are used for the other layers of a knowledge-based system (see introduction). Sometimes a case tool is used specifically designed for developing knowledge based systems like PC-Pack, but the support for specifying other layers than the knowledge layer is minimal.

Currently there are at least 30 software packages for configuration, 120 for planning and scheduling, 20 for diagnosis, a number of packages for plain matching etcetera. These solutions are for so-called horizontal markets. For vertical markets, like retail, the stock market or air traffic transport also more or less standard software enters the market. Naturally these solutions have to be tuned to their specific use and application and to the environment they should be implemented in. There will be always room for tailored solutions, but the market share of tailored solutions will one day drop in favour of standard solutions.

Knowledge-based technology has become an accepted technology. Different and more severe requirements count nowadays when developing knowledge-based solutions compared to the early days. What started from laboratory systems developed into full production systems supporting up to thousands of users and batchjobs and supporting mission-critical industrial and space travel processes. The development environments have a corresponding reliability.

6 REFERENCES

Akkermans, J.M., Speel, P.H. and Ratcliffe, A. (1999). Problem, Opportunity, and Feasibility Analyusi for Knowledge Management - An Industrial Case Study. In *Proceedings of the 12th Banff Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff, Canada, October 16-21, 1999

Angele, J., Fensel, D., Landes, D., and Studer, R. (1998). Developing knowledge-based systems with MIKE. *Journal of Automated Software Engineering*

Anjewierden, A., Wielemaker, J. and Toussaint, C. (1990). SHELLEY: Computer Aided Knowledge Engineering, in B. Wielinga, J. Boose, B. Gaines, G. Schreiber & M. van Someren (Eds.), *Current Trends in Knowledge Acquisition*, Amsterdam: IOS Press

Benjamins, V.R. 1993. *Problem Solving Methods for Diagnosis.* Ph.D. thesis, University of Amsterdam, Amsterdam

Benjamins, V.R., Fensel, D. and Straatman, R. (1996). Assumptions of problem solving methods and their role in knowledge engineering. In W. Wahlster, ed. *Prodceedings ECAI-96*, pp. 408-412

Benjamins, V.R. and Fensel, D. (1998). Special Issue Problem Solving Methods. *International Journal of Human-Computer Studies,* 49(4): 305-650

Breuker, J.A. and Van de Velde, W. editors (1994). *The CommonKADS Library for Expertise Modelling.* Amsterdam, IOS Press

Booch, G., Rumbaugh, J. and Jacobson, I. (1998). *The Unified Modelling Language User Guide*. Reading, MA, Addison-Wesley

Checkland, P. and Scholes, J. (1990). *Soft Systems Methodolody in Action.* Chichester, UK, Wiley

Clancey, W.J. (1985). Heuristic classification. *Artificial Intelligence,* 27:289-350

Davenport, Thomas H. & Prusak, Laurence (1997); *Working Knowledge: How Organizations Manage What They Know*, Harvard Business School Press

D'Souza, D.F. and Wills, A.C. (1998). Objects, *Components and Frameworks with UML: the Catalysis Approach*, Reading, MA, Addison-Wesley

Eriksson, H.-E and Penker, M. (1998). *UML Toolkit*. New York, Wiley

Gaines, B.R. (1997). Special Issue on Using Explicit Ontologies in KBS Development, *International Journal of Human-Computer Studies*, Vol. 46, No. 2/3, Jun 1997

Gaines, B.R., Musen, M.A. and Uthurusamy, R. (1997): Artificial Intelligence in Knowledge Management, Papers from the 1997 AAAI Symposium, Technical Report SS-97-01, AAAI Press, Menlo Park, 1997

Hayes-Roth, F., Waterman, D.A., and Lenat, D.B. (1983). *Building Expert Systems.* New York, Addison-Wesley

Jacobson, I., Booch, G., and Rumbaugh, J. (1999). *The Unified Software Development Process.* Addison Wesley Longman

Johansson, H.J., McHugh, P., Pendlebury, A.J., and III, W.A.W. (1993). *Business Process Reengineering.* New York, Wiley

McGraw, K.L. & Harbison-Briggs, K. (1989). *Knowledge Acquisition: Principles and Guidelines*, Englewood Cliffs, NJ, Prentice-Hall

McGraw, K.L. & Harbison, K. (1997). *User-Centered Requirements: The Scenario Based Engineering Process,* Lawrence Erlbaum Associates

Motta, E. (1999). *Reusable Components for Knowledge Modelling: Case Studies in Parametric Design Problem Solving*. Amsterdam: IOS Press

Musen, M. (1993). An overview of knowledge Acquisition. In J.M. David, J.P. Krivine and R.Simmons, eds. *Second Gereration Expert Systems*. Berlin: Springer Verlag

Nonaka, Ikujiro & Takeuchi, Hirotaka (1995); *The Knowledge-Creating Company*, Oxford University Press, New York

O'Hara, K., Shadbolt, N.R. & van Heijst, G. (1998). Generalized directive models: integrating model development and knowledge acquisition, *International Journal of Human-Computer Studies*, Vol. 49, 497-522

Orsvärn, K. (1996a). Principles for Libraries of Task Decomposition Methods -- Conclusions from a Case-Study. In Nigel Shadbolt, Kieron O'Hara & Guus Schreiber, editors, *Advances in Knowledge Acquisition*, pages 48--65. Springer-Verlag, Berlin Heidelberg, Germany

Orsvärn, K. (1996b). Knowledge Modelling with Libraries of Task Decomposition Methods. Ph.D. Thesis, Swedish Institute of Computer Science, Sweden

Puppe, F. (1993). *Systematic introduction to Experts Systems. Knowledge Representation and Problem Solving Methods*. Berlin. Springer Verlag

Schreiber, A.Th., Akkermans, J.M., Anjewierden, A.A., de Hoog, R., Shadbolt, N.R., Van de Velde, W., and Wielinga, B.J. (2000). *Knowledge engineering and management: The CommonKADS methodology*. Cambridge Massachusetts, MIT Press

Scott-Morgan, P. (1994); *TheUunwritten Rules of the Game*. New York, McGraw-Hill

Scott, A.C., Clayton, J.E. and Gibson, E.L. A practical guide to knowledge acquisition

Shadbolt, N.R. & Burton, A.M. (1990); Knowledge elicitation. In J.R. Wilson & E.N. Corlett, Eds., *Evaluation of Human Work: A Practical Ergonomics Methodology*, Taylor and Francis, London, 321-345

Shadbolt, N.R. & Milton, N. (1999). From knowledge engineering to knowledge management, to appear in a special BAM'98 (British Academy of Management) conference issue of the *British Journal of Management*, spring 1999

Shadbolt, N.R., O'Hara, K. & Crow, L. (forthcoming); The Experimental Evaluation of Knowledge Acquisition Techniques and Methods: History, Problems and New Directions, *International Journal of Human-Computer Studies*

Speel, P.H. & Aben, M. (1997); Applying a library of problem-solving methods on a real-life task, *International Journal Human-Computer Studies*, 46: 627-652

Speel, P.H. & Aben, M. (1998); Preserving conceptual structures in design and implementation of industrial KBSs, *Special issue on Problem-Solving Methods for the International Journal of Human-Computer Studies*, 49: 547-575

Speel, P.H., Shadbolt, N.R., Vries, W. de, Dam, P.H. van, O'Hara, K. (1999). Knowledge Mapping for Industrial Purposes. In *Proceedings of the 12th Banff Knowledge Acquisition for Knowledge Based Systems Workshop*, Banff, Canada, October 16-21, 1999

Tu, S.W., Eriksson, H., Gennari, J.H., Shahar, Y., and Musen, M.A. (1995). Ontology-based configuration of problem-solving methods and generation of knowledge acquisition tools: The application of PROTÉGÉ-II to protocol-based decision support. *Artificial Intelligence in Medicine*, **7**(5)

Vail III, Edmond F. (1999); Mapping Organizational Knowledge, *Knowledge Management Review*, Issue 8, May/June 1999, 10-15

Van Joolingen, W., Schreiber, A. Th., and Hermans, L. (1999). *Handbook SDF-II*. Utrecht, The Netherlands, CIBIT

Von Krogh, G., Nonaka, I., and Ichijo, K. (1997). Develop knowledge activists! *European Management Journal*, 5: 475-483.

Von Krogh, G., Nonaka, I., and Nishiguchi, T. (2000). *Knowledge Creation: A New Source of Value.* London, MacMillan, forthcoming

Watson, G. (1994). *Business Systems Engineering.* New York, Wiley

Wielinga, Bob, Sandberg, Jacobijn & Schreiber, Guus (1997); Methods and Techniques for Knowledge Management: What Has Knowledge Engineering to Offer?, *Expert Systems With Applications*, Pergamon, Elsevier Science Ltd, 1997, Volume 13, Number 1: 73-84

Wigg, Karl M., Robert de Hoog & Rob van der Spek (1997); Supporting Knowledge Management: A Selection of Methods and Techniques, *Expert Systems With Applications*, Pergamon, Elsevier Science Ltd, 1997, Volume 13, Number 1: 15-27