Int. J. Human–Computer Studies (1997) **45,** 183–292

Using explicit ontologies in KBS development

G. van Heijst,* A. Th. Schreiber and B. J. Wielinga
University of Amsterdam, Department of Social Science Informatics, Roetersstraat 15, NL-1018 WB, Amsterdam, The Netherlands.
email: gertjan/schreiber/wielinga@swi.psy.uva.nl

This article presents a number of ways in which ontologies—schematic descriptions of the contents of domain knowledge—can be constructed and can be used to improve the knowledge engineering process. The main message is that early in the knowledge engineering process an application-specific ontology should be constructed. To facilitate this, the article presents some principles for organizing a library of reusable ontological theories which can be configured into an application ontology. This application ontology is then exploited to organize the knowledge acquisition process and to support computational design. The process is illustrated with a knowledge engineering scenario in the domain of treating acute radiation syndrome.

1. Introduction

During the last decade, comprehensive knowledge-engineering methodologies have emerged which provide support for organizing the development process of knowledge-based systems. Examples are the Generic Task approach (Chandrasekaran, 1987), the Role-Limiting Methods approach (McDermott, 1988), the Components of Expertise approach (Steels, 1990), the KADS methodology (Wielinga, Schreiber & Breuker, 1992) and the Protégé framework (Musen, 1989*b*). These approaches share the characteristic that they promote the reuse of knowledge elements by providing libraries of off-the-shelf knowledge components. Such libraries are necessary to turn knowledge engineering from an ''art'' into a proper engineering discipline. So far, the emphasis has mainly been on *problem-solving methods*—abstract descriptions of the steps that must be taken to perform particular tasks.

Another type of knowledge which has been suggested as a candidate for reuse are *ontologies*—intensional descriptions of the domain knowledge in some field. Many researchers feel that access to libraries of reusable ontological components would facilitate the knowledge engineering process and several research groups have taken up the challenge of developing candidate components. However, the field is still in its infancy and many problems are unsolved or even unaddressed. To mention a few: how can ontologies be built, compared, integrated, validated, visualized or used?

In addition, the question needs to be addressed whether a methodology that is based on the use of generic problem-solving methods can also be based on the use of generic components for ontologies. In other words: can library components be specified in such a way that ontologies can be used with different problem-solving methods and vice versa?

* Present address: Kenniscentrum Cißit, Arthut van Schendelstraat 570, 3500 AN Utrecht, The Netherlands.

1071-5819/97/2130183 + 110$25.00/0/hc960090

This question touches upon a long-standing debate in AI about whether domain knowledge can be represented independently of how it is used in reasoning. Clancey's early work on NEOMYCIN suggested that both domain knowledge and problem-solving knowledge can be reused, provided that the problem-solving knowledge and domain knowledge are represented separately in the knowledge base (Clancey & Letsinger, 1984). This belief that separation of control knowledge and domain knowledge would enhance the reusability of both was also one of the assumptions that led to the conception of the KADS four-layer model (Wielinga & Breuker, 1986). However, Bylander and Chandrasekaran (1988) argued against this belief by presenting the *interaction problem*:

> Representing knowledge for the purpose of solving some problem is strongly affected by the nature of the problem and the inference strategy to be applied to the problem (Bylander & Chandrasekaran, 1988).

The interaction problem states that the ontology of the knowledge in a KBS is strongly affected by the task of the KBS and the methods it uses to perform that task. Bylander and Chandrasekaran identified two reasons for the interaction problem. Firstly, the application task determines to a large extent which kinds of knowledge should be encoded. In general, it is not feasible nor desirable to model everything the expert knows. Secondly, the knowledge must be encoded in such a way that the inference strategy used can reason efficiently.

In this article we study the general question of how (explicit) ontologies can be obtained and used to make the knowledge-engineering process more manageable. In this context we address a number of relevant research issues. Firstly, we consider the way in which the knowledge-engineering process needs to be organized in order to make explicit ontologies useful. In order to use ontologies profitably in knowledge engineering, they must be embedded in a methodology. In Section 2 an overview is presented of the way in which current knowledge engineering approaches organize the KBS development process. The role of ontologies in this process is analysed. A second issue concerns the way ontologies are obtained. Basically, there are three ways: ontologies can be constructed from scratch, they can be selected from a library of off-the-shelf ontologies, or they can be configured from off-the-shelf components. This issue is addressed in Sections 3 and 7. Thirdly, we study the various ways in which an ontology can be exploited to support the knowledge engineering process and to improve the quality of the resulting knowledge based system. (Sections 4–6). Finally, the relationship between ontologies and problem-solving methods needs to be studied. It is important to get a handle on the interaction problem to maximize reuse of both ontologies on problem-solving methods.

2. The knowledge engineering process

To determine how explicit ontologies can be used in knowledge engineering, we must have an understanding of how the knowledge engineering *process* is organized. During the last decade, a number of approaches to knowledge engineering were proposed that are similar in spirit, although they differ in their details and terminology. This section presents an overview of the characteristics that are shared by these approaches. In the sections that follow we will argue how explicit ontologies can be useful within this general paradigm.

A first characteristic shared by current approaches is that they view knowledge engineering as a *modelling* process, as opposed to the older "mining" view. Knowledge engineering is a creative activity which can be supported by providing *modelling principles.* A second characteristic is that the resulting knowledge models should be formulated at the *knowledge level* (Newell, 1982). Knowledge level models emphasize the rational behind problem solving in terms of goals, actions and knowledge; they abstract away from how these are implemented in specific representation formalisms. The actual implementation of the problem solving competence in a knowledge based system is delineated in a second model, the *design model.* In the design model additional decisions are taken which enable a computer system to realize the problem solving competence in an efficient way. The design process can be supported by means of *design principles.*

2.1. KNOWLEDGE MODELLING PRINCIPLES

In an overview of the field, Musen and Schreiber (1995) identify three modelling principles that lie at the heart of all recent knowledge engineering approaches. These are the *role-limiting* principle, the *knowledge typing* principle and the *reusability* principle. Each will be described briefly. In addition, we consider the use of *skeletal models* as a fourth general knowledge engineering principle.

2.1.1. Role-limiting

Role limiting is a mechanism for organizing knowledge by putting constraints on the ways knowledge elements of particular *types* can be used in reasoning. Wielinga, Van de Velde, Schreiber and Akkermans (1993) formulate the role-limiting principle as follows.

> An intelligent agent which is faced with a particular task can be modeled as imposing on its knowledge a structure, the parts of which play different, specialized and restricted roles in the totality of the problem-solving process.

2.1.2. Knowledge typing

The role-limiting principles states that different knowledge elements play different roles in reasoning. Therefore, knowledge elements must be typed according to their role in problem solving. In the literature, at least five different types of knowledge are distinguished.

- *Tasks* correspond to the goals that must be achieved during problem solving.
- *Problem-solving methods* are ways to achieve the goals described in tasks. In some knowledge modelling frameworks, problem-solving methods define sub-tasks to which other problem-solving methods can be applied. We will call such a decomposition a task instance.
- *Inferences* describe the primitive reasoning steps in the problem-solving process. Inferences are also called *mechanisms.* Together, the inferences form a functional model which is sometimes called the *inference model* or *inference structure.*

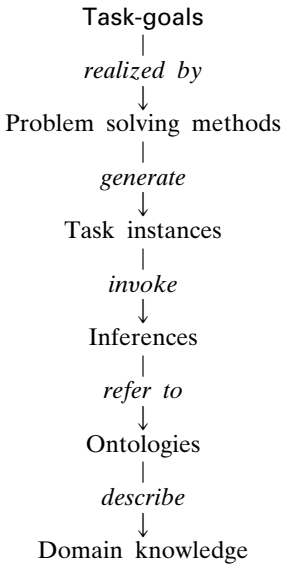

FIGURE 1. The different components of knowledge models.

- *Ontologies* describe the structure and vocabularly of the static domain knowledge.
- *Domain knowledge* refers to a collection of statements about the domain.

Figure 1 shows how the different knowledge model components are related.

2.1.3. Reusability

Current approaches to knowledge engineering emphasize the reuse of knowledge components across domains and tasks. The availability of libraries of validated and well-documented knowledge components not only speeds up the KBS development process but it also facilitates maintenance and upgrading. However, there are differences between the approaches with respect to the nature and the grain size of the components that they consider potentially reusable.

2.1.4. Use of skeletal models

Knowledge model components are often reused in the form of skeletal models. Such models specify one part of a knowledge model (e.g. the problem solving method). The knowledge engineer then has to fill in the other parts to complete the knowledge model. As a result of knowledge typing, the already specified parts in the skeletal model constrain how the other parts can be modelled. This way, skeletal models structure the knowledge modelling process. In the literature one can find skeletal models based on problem solving methods (e.g. Marcus, 1988), inference models (e.g. Breuker *et al.,* 1987) and ontologies (e.g. Musen, Fagan, Combs & Shortliffe 1988).

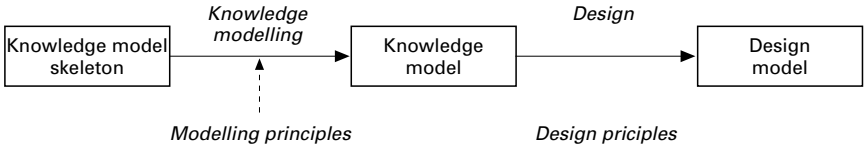

FIGURE 2. A schematic overview of how modern knowledge engineering approaches view the knowledge engineering process.

Figure 2 shows how the models, the activities and the principles are related. Typically, knowledge modelling starts with the selection of a modelling skeleton. This can either be a general modelling framework or a partially instantiated knowledge model. Then the skeleton is completed. This process is guided by the modelling principles. In the design phase, which is briefly described in Section 2.3, a KBS is designed which operationalizes the problem solving competence specified in the knowledge model.

2.2. THE MODELLING PROCESS

In some approaches (e.g. CommonKADS) knowledge engineering starts with modelling the context in which a KBS will function. Such an organization model is useful for requirements analysis and feasibility studies and its construction typically precedes knowledge modelling. For the present purpose, we assume that this has all been done and we concentrate on knowledge modelling.

In the above sections we have briefly described the components of knowledge models. We will now shift our focus to the *process* of devising the knowledge model. In most of the current approaches, constructing a knowledge model involves four activities as follows.

(1) Construct a task model for the application.
(2) Select and configure appropriate ontologies, and if necessary refine these.
(3) Map the application ontology onto the knowledge roles in the task model.
(4) Instantiate the application ontology with domain knowledge.

2.2.1. Constructing a task model for the application

The first activity in KBS construction is task analysis. The purpose of the task analysis is to decompose the real-life task into a number of generic tasks and to associate these with appropriate problem solving methods. Together, the methods and tasks form a task model.

2.2.2. Selecting and configuring an application ontology

When the task of the target system is recognized as a generic-task instance or a sequence of generic-task instances, the next activity involves the construction of an application-specific ontology. In general, ontology construction is a difficult process that requires the expertise of a knowledge engineer or an informed domain expert.

A library of reusable ontological theories can ease this process. The knowledge engineer can select the reusable theories and, if necessary, tune them to meet the demands of the application.

2.2.3. Mapping the task model onto the application ontology

The application ontology defines the relevant concepts in the domain. When performing a generic task, instances of particular concept typically fulfill particular roles in problem solving. For example, in medical diagnosis, instances of the concept disease will often play the role of hypotheses. By defining mappings between the roles in the task model and the concepts in the ontology it is made explicit which concept instantiations may play which roles. The mapping is specific for tasks and domains as is illustrated by the fact that in therapy planning diseases will often play the role of data.

2.2.4. Instantiating the application ontology

While the application ontology defines which concepts are used in the domain, the application knowledge describes the actual instances of these concepts. Besides the reusability aspect, one of the main arguments for distinguishing between the ontology and the application knowledge is that the application knowledge must by definition be presented by the medical expert.

2.3. DESIGN

The knowledge model is an implementation-independent description of the knowledge and methods needed to perform a task. The design model describes how the knowledge model can be operationalized in a knowledge based system. The design model specifies both the general architecture of the KBS and the representations and algorithms that are used by the KBS to perform its task. When developing the design model, the knowledge engineer must take additional decisions to ensure that the KBS is able to perform its task efficiently.

The design process is guided by design principles. To a large extent, these principles are similar to principles for system design in software engineering, such as the use of libraries of reusable software modules. In knowledge engineering, one typically finds this kind of reuse in the form of expert-system shells, which contain reusable reasoning engines. Besides the software-engineering principles there are also principles which are typical for KBS design. An example of such a principle is that of *structure preserving design,* which implies that the information content and structure of the knowledge model is preserved in the final artifact. This principle is derived from the requirement that knowledge based systems must be able to explain their lines of reasoning in expert-understandable terminology. Since the vocabularly of the experts is laid down in the knowledge model, the KBS can only provide this kind of explanations if the information in the knowledge model is also available in the design model. Besides explanation, structure preserving design also facilitates maintenance and code-reuse (Schreiber, 1993).

While current knowledge engineering approaches share many characteristics with respect to knowledge modelling, there is less consensus with respect to the KBS design process. In some knowledge engineering approaches, the design model is

derived automatically from the knowledge model. For example, in the PROTÉGÉ-II framework, the knowledge models are automatically translated in CLIPS production rules. In other approaches, KBS design is considered to be a task for the knowledge engineer (e.g. CommonKADS). Here, the design process is viewed as a knowledge-intensive activity where additional information is added to the knowledge model in order to achieve computational adequacy. The pros and cons of these alternatives are discussed in Sections 4 and 6.

2.4. CONTEXT: THE GAMES-II PROJECT

Most of the work reported in this article was performed in the context of GAMES-II, a research project funded by the European Union. The purpose of this project was to develop methods and tools for medical KBS development. The approach taken in GAMES-II fits well in the general paradigm described above. Because some of the issues discussed in later sections are phrased in GAMES-II-specific terminology, we will now describe some aspect of the GAMES-II work in more detail.

Within the GAMES-II project we have only investigated three types of medical tasks: diagnosis, therapy planning and patient monitoring. Further, it was assumed that these tasks could be modelled by means of a single inference model: the STModel (Ramoni, Stefanelli, Barosi & Magnani, 1992). This model, which is shown in Figure 3, views problem solving as a cyclic process of data abstraction, abductive hypotheses generation and subsequent testing of these hypotheses by means of

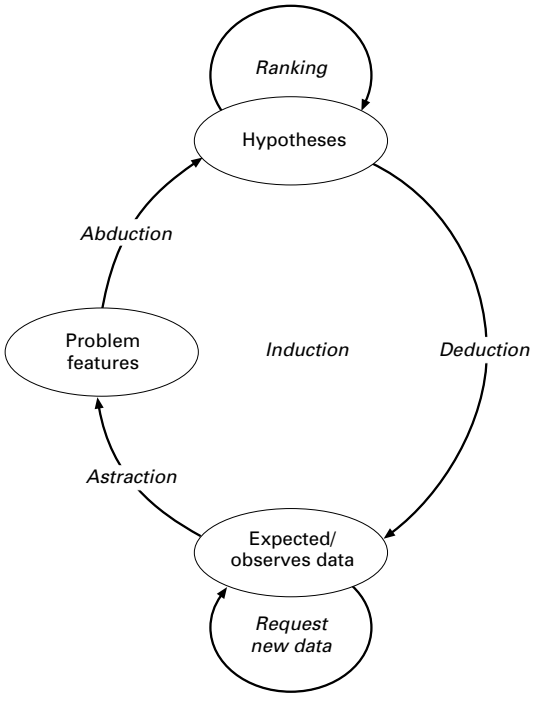

FIGURE 3. Generic inference model (STModel) used in GAMES-II. The arcs in the figure represent inferences, the elipses represent knowledge roles.

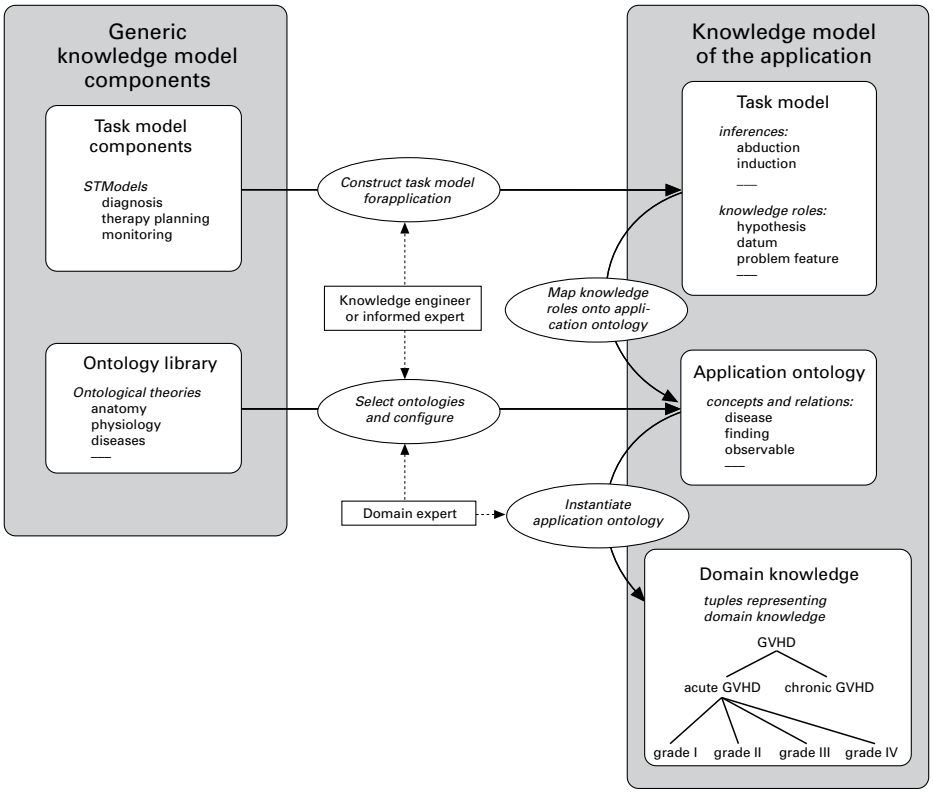

FIGURE 4. Activities in the construction of the knowledge model.

deduction and induction. The arcs in the figure represent inferences; the ellipses are knowledge roles. The order in which the inferences in the task model are executed depends on the problem-solving method used.

To model ontologies, a library of medical ontological theories was developed in GAMES-II. This library is described in Section 3. The library contains definitions of often-found medical concepts such as ''physiological process'', ''therapy'', ''symptom'', etc. Together, the generic tasks and the ontological theories form the reusable knowledge modelling components provided by the project. Figure 4 shows how the GAMES-II-supplied generic knowledge model components support the activities in Section 2.2.

3. Principles for ontology library construction

This section presents principles for organizing a library of reusable ontological theories in the medical field. The focus is on the internal structure of such a library, how it can be built and how it can be used. The proposed principles are illustrated with a library of medical ontologies developed by Sabina Falasconi and described in (Falasconi & Stefanelli, 1994).

In our view, there are two impediments that hinder the development of libraries of reusable ontologies: the *hugeness problem* and the *interaction problem*. The

hugeness problem concerns the overwhelming amount of knowledge in the world. This makes the construction of a library of reusable domain ontologies a daunting exercise. The interaction problem, which was quoted in Section 1, states that domain knowledge cannot be represented independently of assumptions of how it will be used in reasoning.

We will put forward a number of hypotheses about the nature of medical domain knowledge from which principles are derived for organizing a library in such a way that the hugeness problem and the interaction problem remain manageable. In short, these principles are that (i) there is a relatively small set of basic concepts that are reusable across many medical domains and tasks, (ii) medical sub-domains have domain-specific concepts that are often specializations of the basic medical concepts, and (iii) many problem-solving methods require additional concepts that are specific for that method.

The section is organized as follows. In Section 3.1, we present a definition of ontology and a classification of different types of ontologies. Section 3.2 describes the organizational principles that the library is based on, thereby showing how the hugeness problem and the interaction problem can be addressed. Section 3.3 shows how these principles are used to build an initial library and Section 3.4 explains how the library can be used during KBS development.

3.1. ONTOLOGY

In philosophy, the term ''ontology'' refers to ''a particular theory about the nature of being or the kinds of existence''. This broad definition can be interpreted in a number of ways, depending on the metaphysical stance that one takes with respect to what ''existence'' is. A number of researchers in knowledge engineering have therefore suggested more specific, AI-oriented definitions of ontology. In general, AI definitions avoid referring to reality, but rather use terms as representation and conceptualization. An often-cited definition is that of Gruber (1994):

> An ontology is an explicit specification of a conceptualization. The term is borrowed from philosophy, where an ontology is a systematic account of Existence. For AI systems, what ''exists'' is that which can be represented.

Although not explicitly stated, this definition suggests that an ontology is a meta-level description of a knowledge representation. Thus, the ontology is not part of the representation itself. As we shall see in detail in Section 6, this is an aspect of ontologies that will turn out to be important for their application in knowledge engineering. Another aspect of ontology that is important for the work reported here can be found in a definition formulated by Wielinga and Schreiber (1993):

> An (AI-) ontology is a theory of what entities can exist in the mind of a knowledgeable agent.

This definition emphasizes that we want to apply the notion of ontology to all

knowledgeable agents, including humans. Since different knowledgeable agents will often have different symbol-level representations, it is convenient to formulate ontologies at the knowledge level. This aspect is important for knowledge engineering; in Section 5 it will be argued that ontologies can be used as mediators between knowledge as it is understood by a domain expert and knowledge as it is represented in a computer system. A third knowledge-engineering oriented definition of ontologies is given by Alberts (1993):

> An ontology for a body of knowledge concerning a particular task or domain, describes a taxonomy of concepts for that task or domain that define the semantic interpretation of the knowledge.

In AI, the term ontology is often used as a synonym for the terminology in some domain. This definition emphasizes that it is not the terminology itself that constitutes the ontology but the semantic interpretation of the terms. Another important aspect of this definition is that ontologies can be specific for tasks or for domains. That is, both the domain and the task at hand may affect the ontology.

The three definitions above are not contradictory, and capture a large proportion of the aspects of ontology that are relevant for the work described here. Combining the above definitions results in the following definition.

> An ontology is an explicit knowledge-level specification of a conceptualization, i.e. the set of distinctions that are meaningful to an agent. The conceptualization—and therefore the ontology—may be affected by the particular domain and the particular task it is intended for.

Ontologies can be classified according to two dimensions: the amount and type of structure of the conceptualization and the subject of the conceptualization. With respect to the first dimension we distinguish three categories.

- *Terminological ontologies* such as lexicons, specify the terms that are used to represent knowledge in the domain of discourse. An example of such an ontology in the medical field is the semantic network in UMLS (Unified Medical Language System; Lindberg, Humphreys & McCray, 1993).
- *Information ontologies* which specify the record structure of databases. Database schemata are an example of this class of ontologies. Level 1 of the PEN & PAD model (Rector, Nowlan, Kay, Goble & Howkins, 1993), a framework for modelling medical records of patients, is a typical example of such an ontology in the medical field. At this level, the model provides a framework for recording the basic observations of patients, but it makes no distinction between symptoms, signs, treatments, etc.
- *Knowledge modelling ontologies* specify conceptualizations of the knowledge. Compared to information ontologies knowledge modelling ontologies usually have a richer internal structure. Further, these ontologies are often tuned to a particular use of the knowledge that they describe. Within the context of KBS development, knowledge modelling ontologies are the ontologies that we are mostly interested in. The level 2 description of the PEN & PAD model is an

example of a knowledge modelling ontology in the medical field. At this level, the level 1 observations are grouped to describe the decision-making process.

The other dimension on which ontologies can be differentiated is the subject of the conceptualization. Four categories can be distinguished on this dimension: (i) application ontologies, (ii) domain ontologies, (iii) generic ontologies and (iv) representation ontologies.

- *Application ontologies* contain all the definitions that are needed to model the knowledge required for a particular application. Typically, application ontologies are a mix of concepts that are taken from domain ontologies and from generic ontologies (which are described below). Moreover, application ontologies may contain method- and task-specific extensions. Application ontologies are not reusable themselves. They may be obtained by selecting theories from the ontology library, which are then fine-tuned for the particular application. We use the term application ontology in a similar way as in PROTÉGÉ-II (Tu, Eriksson, Gennari, Sharar & Musen 1995).
- *Domain ontologies* express conceptualizations that are specific for particular domains. As mentioned in Section 2, current knowledge engineering methodologies make an explicit distinction between *domain ontologies* and *domain knowledge.* Whereas the domain knowledge describes factual situations in a certain domain, the domain ontology puts constraints on the structure and contents of domain knowledge.
- *Generic ontologies* are similar to domain ontologies, but the concepts that they define are considered to be generic across many fields. Typically, generic ontologies define concepts like state, event, process, action, component, etc. The concepts in domain ontologies are often defined as specializations of concepts in generic ontologies. Of course, the borderline between generic ontologies and domain ontologies is vague, but the distinction is intuitively meaningful and is useful for building libraries.
- *Representation ontologies* explicate the conceptualizations that underly knowledge representation formalisms (Davis, Shrobe & Szolovits, 1993). They are intended to be *neutral* with respect to world entities (Guarino & Boldrin, 1993). That is, they provide a representational framework without making claims about the world. Domain ontologies and generic ontologies are described using the primitives provided by representation ontologies. An example in this category is the *Frame Ontology,* which is used in Ontolingua (Gruber, 1993).

3.2. ORGANIZATION OF THE LIBRARY

This section presents structuring principles for organizing an ontology library and illustrates these principles using a medical ontology library that was developed as a case study in the context of the GAMES-II project. In terms of the categories distinguished in the previous section, the library consists of domain ontologies and generic ontologies of the knowledge modelling type. The domain ontologies in this library are described in (Falasconi, 1993). Many of the generic ontologies were taken from the Ontolingua library developed at Stanford University.

The classification of ontologies presented in Section 3.1 is too coarse-grained to be used as an indexing scheme for the library. Therefore, a number of principles were formulated that allow a more fine-grained categorization. In short, these principles are that (i) there are some general categories of medical knowledge that are fundamental to all kinds of medical reasoning, (ii) in many application domains there are additional ontological distinctions that are specific for that domain, and (iii) the use of specific reasoning methods may require additional method-specific ontological distinctions. Based on these principles, the library is partitioned into two regions: a core library and a peripheral library. The core part contains definitions of the generic concepts and of general medical categories. The peripheral part contains definitions of the domain- and method-specific extensions. The division is important because the two parts are indexed in different ways. Section 3.2.2 describes the core library and in Section 3.2.3 the peripheral parts are explained. Before turning to a more elaborate description of these parts, first some general issues in library construction are addressed.

3.2.1. Issues in library construction

Language. Ontologies need to be specified in a language. A number of languages have been proposed as candidates (e.g. MODEL—Tu *et al.,* 1995; CML—Schreiber, Wielinga, Akkermans, Van de Velde & Anjewierden, 1994), but it is not entirely clear to date which requirements a language for ontological modelling should satisfy. The library presented here is developed with Ontolingua (Gruber, 1993). An Ontolingua ontology consists of a number of *definitions,* collections of labelled sentences that constrain the use of a term. Four kinds of definitions are distinguished: classes, relations, functions and instances. Definitions can be grouped into *theories,* collections of definitions that are somehow related. Theories can include other theories, which means that all the definitions in the included theory are also available in the including theory. Thus, the theory is the main modularity construct available, and is therefore the principal building block of the library that is described below.

Modularity. A key to successful library organization is modularity. A modular organization is one that organizes units in modules so that the cohesion within modules is maximal, while the interaction between modules is minimal. In the ontology library presented in this section, the units are definitions and the modules are theories. There are numerous possible cohesion criteria. Which of these are useful in this context depends on the intended use of the library.

The main intended use of the library is to support the construction of application ontologies. Therefore, definitions that are likely to be used in the same application ontologies should be put together into one theory. There are two features that determine which definitions are needed for an application ontology: (i) the (medical) *sub-domain* that the application should reason about and (ii) the *method* that the application uses to perform a (sub-)task. For example, applications in the domain of cardiac diseases use (at least partially) other knowledge than that used by applications in the domain of bacterial diseases; similarly, applications that *diagnose* cancer are likely to use different knowledge from applications that *plan* cancer therapy.

Alternative definitions. It is important to stress that the library is not intended as *the* ontology of medical knowledge; the definitions are not claimed to capture the essence of knowledge categories in some Platonic sense. Instead, the definitions should be viewed as conceptualizations that have been proven useful for solving medical problems, either by human experts or by computers. A consequence of this pragmatic point of view is that it is sometimes necessary to allow for alternative, or even inconsistent, definitions of a concept in the library. For example, an often used concept in medical reasoning is ''causality''. Since this concept is reusable across many applications, it is an obvious candidate for inclusion in the library. However, the history of philosophy shows that is is extremely difficult to come up with a satisfying definition of causality. When we look at medical reasoning, it seems that a number of alternative conceptualizations are being used. For example, in some cases both the cause and the effect roles of the `causes` relation are constrained to be physiological states, while in other cases they need to be events. The temporal aspects of the concept may also vary; in some cases the relation between cause and effect is immediate, while in others there may be a delay. Because these alternative conceptualizations are useful in medical reasoning, we have chosen to allow multiple definitions of the same concept, leaving the decision of which conceptualization is appropriate in a particular context to the library user.

The need for a higher-order language. The requirements of a modular organization and multiple concept definitions make it necessary to allow higher-order expressions in the ontology specification. The principle of modularity requires that the more generic aspects of a concept are defined in a core library theory, while the more domain- or method-specific aspects of those concepts are defined in a more peripheral theory. Take the previous example, assume that in a core theory `causes` is defined as a binary relation that takes states as arguments:

`causes (⟨state1⟩, ⟨state2⟩)`

For some method in some domain, the definition of the causal relation needs to be augmented with a notion of time delay. The typical first-order solution to do this would be to add a third parameter to `causes`:

`causes(⟨state1⟩, ⟨state2⟩, ⟨delay⟩)`

It is clear that the introduction of an extra parameter violates the earlier mentioned minimal interaction principle, and thus the principle of modularity. The addition of the time delay parameter leads to the destruction of the internal structure of the generic definition of `causes`, with the result that all the definitions that rely on the definition of `causes` also need to be updated. To avoid this, the domain- and task-specific specializations must be specified by means of higher-order expressions, such as the following, where `causes-tuple` refers to a tuple in the extension of the causes relation:

`time-delay(⟨causes-tuple⟩, ⟨delay⟩)`

Unfortunately, allowing higher-orders introduces some well known difficulties. Firstly, higher-order languages are not decidable, thus it is impossible to have a

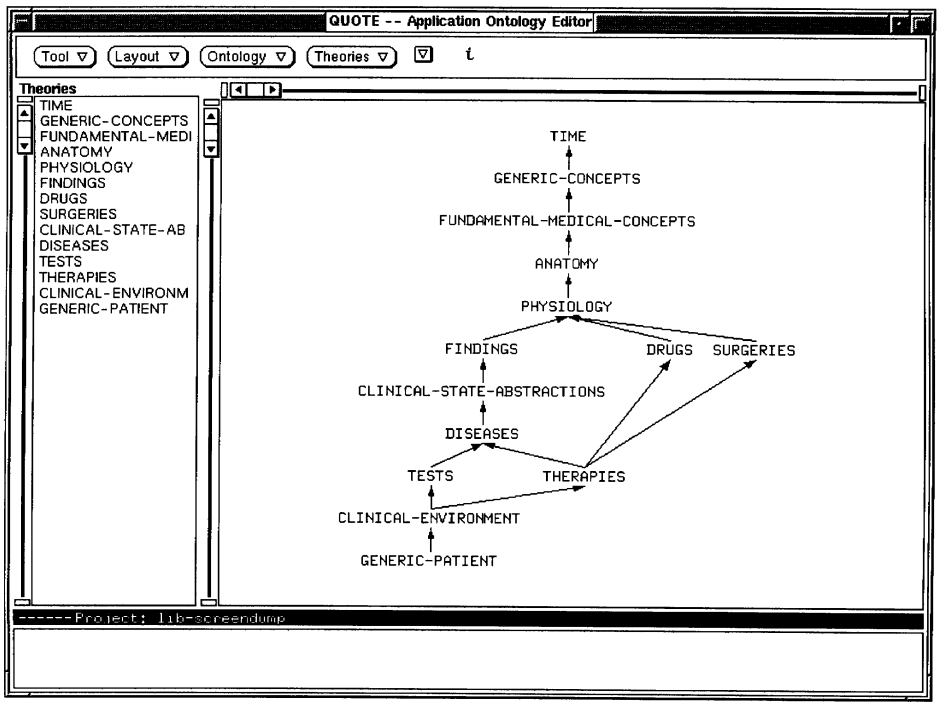

FIGURE 5. Theory inclusion graph of the theories defining the basic categories of medical knowledge. Each node in the graph represents a theory, with its own set of definitions.

system that can prove the internal consistency of the ontological theories. Secondly, the use of a higher-order language introduces the risk of self-referential sentences and the paradoxes that they give rise to. Since the language will be used for library construction, and not for reasoning, we allow the modularity argument to prevail.

3.2.2. Basic categories of medical domain knowledge

This section describes the core part of the library, which contains definitions that are considered reusable across many medical domains and medical tasks. Figure 5 shows a part of the theory structure of this section of the library, in the form of a theory-inclusion graph. The nodes in the graph represent ontological theories, and the edges denote inclusion relations. Each arrow points from an including theory to an included theory. If a theory includes another theory, this means that all the definitions in the included theory are also available in the including theory.

Criteria for partitioning definitions. The decisions about the partitioning of definitions into theories are based on two considerations which we describe further below: (i) the definitions are to be centred around some ''natural categories'', and (ii) the number of theory inclusions must be kept to a minimum.

Centre definitions around natural categories. The main criterion for partitioning the definitions into theories is based on the observation that there are some, but not too many, basic categories of medical knowledge. These categories are natural in the sense of Rosch (1973), in that they reflect a social consensus that exists in the

medical community. Examples of natural categories in the medical domain are concepts such as `patient`, `disease`, `therapy`, etc. These concepts provide a coherent body of terminology that allows medical professionals from different specialities to communicate. These categories recur in almost all medical literature, and they often provide starting points for information analyses for software development.

The natural categories are used as anchor points for modularizing the core library. For instance, the theory diseases is centred around the concept of `disease`, which is represented as an Ontolingua class. On the instances of this class several relations are defined. These definitions, such as `disease-etiology` and `disease-location`, are also located in the diseases theory, since they have no meaning independent of the meaning of disease. The current organization of the domain theories, as shown in Figure 5, is based on the knowledge categories that are distinguished in a number of existing expert systems (e.g. M-KAT; Lanzola & Stefanelli, 1992; and ABEL; Patil, 1981).

Minimization of the number of inclusions. An agent that commits to a particular theory necessarily also commits to the theories included by that theory. Therefore, organizing the theories in such a way that a theory includes few other theories, reduces the overhead of committing to that theory and allows a more flexible use of the library. Therefore, the second criterion used to partition the definitions into theories is that the number of inclusion links must be kept to a minimum. A theory must include, directly or indirectly, the minimal set of theories that it presumes. For example, the concept `disease`, which is defined in diseases, is a sub-class of `clinical-process`, which is defined in `fundamental-medical-concepts`. Therefore, it is necessary that diseases includes fundamental-medical-concepts.

As depicted in Figure 5, two indirect inclusion paths connect clinical-environment, defining concepts related to the context in which medical activities take place, to diseases. The classes `therapy` and `test` are defined in separate theories, enabling external agents to commit to one of the theories without committing to the other. However, because both theories include diseases, all agents committing to one of the two theories must commit to the same definition of diseases. For this reason it is important to avoid ontological overcommitment. In the core part of the library only general characteristics of the concepts should be defined, more specific characteristics should be defined as domain- or method-specific extensions in the peripheral areas of the library.

Contents of the core library. Table 1 contains brief descriptions of some of the theories in the core library which is shown in Figure 5. As an example, Figure 6 shows the Ontolingua definition of the class `observable` which is defined in the theory findings. The sentence labelled as `:axiom-def` expresses that `observable` is a sub-class of `human-body-state-variable`, which is defined in the theory fundamental-medical-concepts. The `:axiom-constraints` sentence defines four possible sub-classes of `observable`. The difference between the `:axiom-def` and `:axiom-constraints` sentences is that the former are considered to be definitional while the latter are assertional (for an explanation of the difference, see Gruber, 1992). The terms `subclass-of` and `subclass-partition` are defined in the Frame ontology.

TABLE 1
Characterization of some theories in the core library as shown in Figure 5 and described in (Falasconi & Stefanelli, 1994)

Theory	Characterization of contents
generic-concepts	Defines basic notions such as `system`, `process`, `action` from an ''engineering'' point of view. For example, a system is conceptualized as a collection of interconnected components characterized by states and processes.
fundamental-medical-concepts	Contains definitions of basic notions useful for medical knowledge representation, such as `human-body` and `medical-agent`. The definitions in this theory specialize notions defined in generic concepts. For example, `human-body` is a sub-class of the class `system`, i.e. it is conceptualized as a class of complex entities describable through states and concerned with physiological or pathological (e.g. clinical) processes.
anatomy physiology	Define ontological categories such as `anatomical-part`, `physiological-process` and `organ` that are generally used in medical contexts. The definitions are mostly based on the work of Patil (1981).
findings drugs surgeries	Define and classify respectively observable findings, conceptualized as values on state variables that indicate the clinical state of a patient, drugs and surgical interventions. They are useful for mapping knowledge modelling ontologies onto ''information ontologies'' underlying the patient medical-record structure.
clinical-state-abstractions	Defines concepts for representing clinical states in compact ways, for instance, to synthesize a set of patient findings. This theory defines, for example, (i) `qualitative-clinical-state-abstraction` expressed using symbolic values such as ''low'' or ''high'', and (ii) `quantitative-clinical-state-abstraction` expressed using numerical values (e.g. a measure such as the body surface computed from body weight and height).
diseases	Defines a `disease` as a clinical process whose evolution can be described through finding or clinical abstraction-values over time, and tries to define taxonomies, used commonly in medical practice, based on diseases characteristics such as time evolution characteristics (e.g. ''acute'', ''chronic''), etiology and location.

The principle behind the core definitions is that these should be minimal. For example, stating that an observable is associated with a quantitative value set (the possible values of the observable are numbers) would be an ontological overcommitment, as this is not likely to hold for every application. Therefore, such a qualification should be defined as an extension.

```
(define-class OBSERVABLE (?observable)
  ''An observable is a state-variable whose values can – contextually – indicate
pathological or physiological states which can be observed. They can be
classified according to the way they are obtained.''
  :AXIOM-DEF (subclass-of observable human-body-state-variable)
  :AXIOM-CONSTRAINTS (subclass-partition observable
                                          (set-of sign
                                                  laboratory-observable
                                                  special-investigation)))
```

FIGURE 6. The Ontolingua definition of the notion of ''observable''. Ontolingua definitions consist of the name of the defined concept, a number of instance variables, and sets of labelled sentences. The sentence labelled as `:axiom-def` defines that `observable` is a sub-class of `human-body-state-variable`, which is defined in fundamental-medical-concepts. The `:axiom-constraints` sentence defines four possible sub-classes of observable. For details of the Ontolingua language, see Gruber (1993).

3.2.3. Method- and domain-specific extensions

The categories described in the previous section are considered basic, in the sense that they are more or less standard across medical tasks and medical domains and form a generally agreed upon body of terminology in the medical field. We have already mentioned that this set of theories, while relatively small, still allows for alternative definitions. In the core part of the library, the definitions are very general, in the sense that they allow for further specialization according to application specific requirements. We will now describe the more application dependent parts of the library. Applications may vary on two attributes: (i) the domains that they reason about, and (ii) the tasks that they perform and the methods that they use.

Reuse of domain-specific concepts across domains. At first glance, the reuse of domain-specific concepts across domains seems a contradiction in terms. However, domain-specificity is not a dichotomy: some concepts are obviously more domain specific than are others. For example, the concept of ''fungal skin infection'' is more specific than that of ''dermatological disease'', while both are more specific than ''disease''.

This observation can be used to organize the library in such a way that more reusable concepts are put in other theories than less reusable concepts. To do this, the notion of domain specificity must be operationalized. One candidate for this operationalization is the notion of abstraction level: definitions that specify less detail are often less domain specific than definitions that specify more detail. However, there are some problems with this operationalization. Firstly, the relation between more abstract and less abstract definitions is a many to many relation. A concept which is specified in detail can have multiple abstractions, depending on the point of view that one takes. This makes it difficult to specify the inclusion relations between theories which contain detailed definitions and theories which contain abstract definitions. In principle, a theory containing detailed definitions should include all the theories that contain abstract definitions of the same concept. However, this would violate the criterion that the number of inclusion relations should be kept to a minimum. A second problem with an organization according to the level of abstraction is that this dimension does not discriminate between concepts on the same level of abstraction. For example, concepts such as ischemia

and glaucoma, which are on the same level of abstraction, are likely to be reusable under different circumstances. An organization along the dimension of abstraction level would not make this explicit.

Issues such as the ones mentioned above make it clear that there are many unsolved problems with respect to the organization of a library in such a way that concepts that are likely to be reused under the same circumstances are stored in the same theory. Therefore we have adopted a pragmatic approach. Taking the division of medical practice as a starting point, every concept in the peripheral part of the library is associated with a *domain-specificity* value. The domain-specificity attribute indicates to what sub-domain, or set of sub-domains, a concept applies. To decide on the domain-specificity of concepts, a hierarchy of medical specialities is used. Each of the nodes in this hierarchy represents a medical sub-domain that may be used as a value for the domain-specificity attribute of a concept. When a concept has a particular domain as its domain-specificity value, it is specific for that domain, but it is reusable across all its sub-domains.

The domain hierarchy reflects the existing organizational structure of medical practice. Example elements of the hierarchy are disciplines such as immunology, pathology, internal medicine and its specializations, etc. Of course, the organization of medical practice varies between countries. Therefore, the structure of the peripheral parts the library is to a certain extent *situated.* This is another motivation for distinguishing between a ''universal'' core library and situated extensions of that core. Figure 7 shows a part of the domain hierarchy.

Reuse of method-specific concepts across methods and tasks. According to the interaction problem, the way in which knowledge is represented is necessarily highly interwoven with the way that knowledge will be used in reasoning. Therefore, it is difficult to reuse knowledge that is defined with a particular method in mind for another method. Taken literally, the interaction problem precludes the reuse of concepts across methods. In this section it will be argued that the interaction problem does not hold to the same extent for every concept, and it will be shown that the degree of *method-specificity* of concepts can be used as an index to organize the ontology library.

As mentioned in Section 2, we have concentrated on three medical tasks:

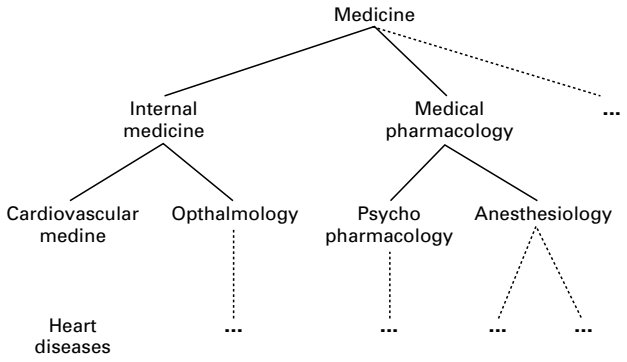

FIGURE 7. A part of the domain hierarchy for the medical field. The hierarchy reflects the organization of the medical practice.

diagnosis, therapy planning and patient monitoring. Furthermore, it is assumed that these generic tasks can be modelled as instantiations of one inference model: the STModel (Figure 3).

Each of the inference steps in the STModel can be realized through a number of methods, and each of these methods may have specific ontological requirements. For example, the abduction of hypotheses from patient findings can be done by interpreting direct associations between findings and diseases. This method thus has the ontological requirement that such associations exist. The following production rule is an illustration of this kind of abduction:

IF chest-pain = present **AND**
  sustained-pain = yes
THEN myocardial-infarct = probable

In some systems that perform abduction by direct associations, the associations are qualified with *certainty factors,* representing the likelihood that the disease is the cause of the findings. This is, for example, the case in MYCIN (Shortliffe, 1979). Using this method thus introduces another ontological requirement.

Alternatively, the diseases that may cause a particular finding could be found by tracing pathways in causal networks—a method which requires the existence of *causal connections* in the domain. For specific methods, the causal links in such networks may need further qualification. For instance, CHECK (Console & Torasso, 1993), a system for abductive diagnosis, makes a distinction between necessary causal connections and possible causal connections. Another example of this is provided by causal-probabilistic networks, where the causal relations are quantified through probability distributions.

Based on the ontological commitments that they require, the methods employed in medical reasoning can be organized in a specialization hierarchy. Descending this hierarchy introduces additional ontological commitments. Figure 8 shows a part of the method hierarchy for abducting diseases from findings in medical diagnosis. The concepts of disease and finding, which are used by all methods for medical abduction, are defined in the core library. The `manifestation-of` relation, which models direct associations between findings and diseases, is specific for methods that are specializations of ''abduction by direct associations between findings and diseases'' (Method 2.1 in Figure 8). Further specializations of these methods may require additional ontological commitments, such as the existence of certainty factors or evoking strengths for these direct associations.

The level of the method hierarchy were an ontological requirement is introduced, is an indicator for the method-specificity of the corresponding concept. In the same way that the domain hierarchy is used to associate concepts with domain-specificity attributes, the method hierarchy is used to assign a method-specificity values to concepts.

It should be emphasized that the organization of methods according to the ontological commitments that they introduce is only one possible way of organizing problem solving methods. For the purpose of the ontology library, this organization is suitable because the hierarchy will be used for retrieving the definitions that are required by the methods. However, we do not claim that we have solved the problem of indexing problem-solving methods.

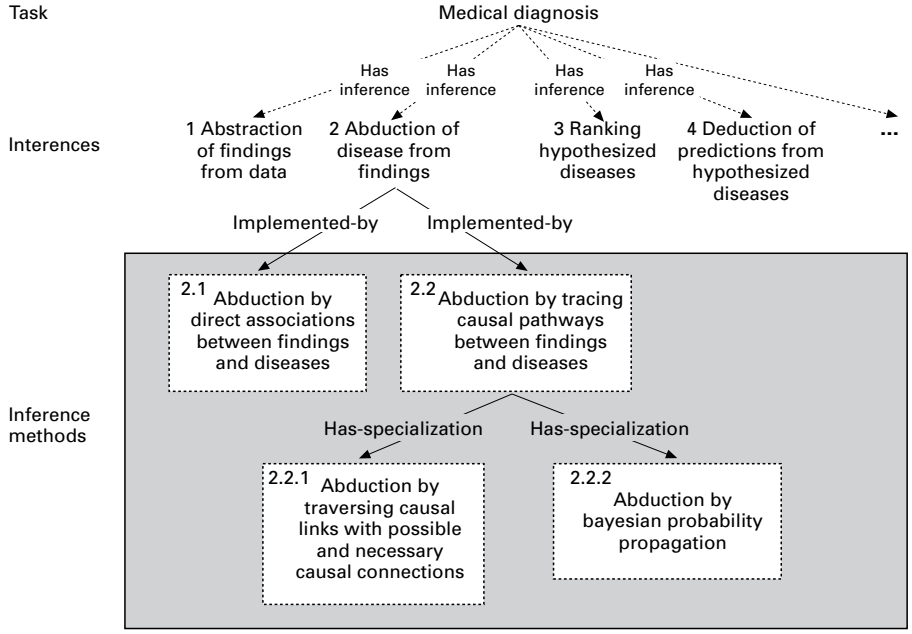

FIGURE 8. Partial hierarchy of methods used in medical diagnosis.

3.2.4. Structure of the library

Section 3.2.2 argued that there are basic categories of medical knowledge that are reusable across all medical domains and medical tasks. These categories form the core part of the library, and they are organized in theories according to the criteria mentioned earlier.

Two attributes determine the degree of reusability of a concept: the domain-specificity and the method-specificity. For the definitions in the core part of the library, these attributes are not discriminating, as they are intended to be reusable across most medical domains and methods. However, this is not the case for the definitions in the extended part. By making the value of concepts on these attributes explicit, it is possible to determine *to what extent* and *under which circumstances* these concepts can be reused. Since concepts that have the same values on both attributes are likely to be applicable under the same circumstances, they should be stored in one theory. In this way the attributes provide a scheme for modularization.

For every combination of a node from the domain hierarchy and a node from the method hierarchy, there can be a theory in the library. This theory contains the definitions that are specific for the method and the domain, but that are reusable across the specializations of the method and sub-domains of the domain.

For instance, the theory ''abduction by tracing causal pathways between findings and diseases in the domain of cardiovascular medicine'' would contain all the definitions that are specific for that method in that domain (e.g. artery-obliteration), but it would not specify that there are probability distributions that describe the nature of the causal connection between pathophysiological states, since these are specific for one particular specialization of the causal tracing method (see Figure 8). The theory would also not contain a definition of pathophysiological state. Since

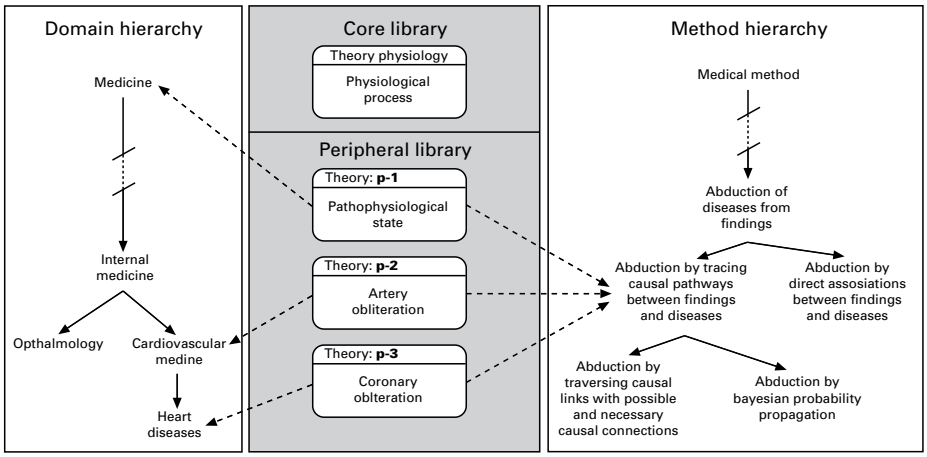

FIGURE 9. The organization of the peripheral parts of the library. The dashed lines show how the theories in the peripheral library are indexed on domain specificity and method specificity. The arrows in the two hierarchies represent specializations. Thus, cardiovascular medicine is a specialization of internal medicine and ''abduction by traversing causal links with possible and necessary causal conditions'' is a specialization of ''abduction by tracing causal pathways between findings and diseases''. Concepts are stored in the theories with the same values on the domain- and method-specificity attributes.

this concept is reusable across a wider range of domains than cardiology, it is defined in the core library.

Figure 9 shows the organization of the peripheral part of the library by example. The dashed arrows in this figure represent the values on the domain-specificity and method-specificity indexes. The arrows in the two hierarchies represent specialization relations. Thus, cardiovascular medicine is a specialization of internal medicine and ''abduction by traversing causal links with possible and necessary conditions'' is a specialization of ''abduction by tracing causal pathways between findings and diseases''. Concepts with the same method specificity and the same domain specificity are stored in the same theory. Retrieving concepts from the library thus amounts to indicating the domain(s) and the method(s) that are relevant for the application and then collecting the theories that have the domain(s) and method(s)—or their parents in the hierarchy—as indexes.

3.3. BUILDING THE LIBRARY

The previous section described the principles of organizing the library of medical ontologies. Here, the issue of filling the library is addressed. Because this involves a large amount of work, only a prototype library has been developed in our project. Rather than aiming at completeness, the project focuses on formulating standardized procedures for adding new definitions to the library. The availability of standardized procedures will make it easier to augment the library and it will enable the development of tools for semi-automatic library maintenance. The currently used procedure consists of four steps: (i) take an existing medical AI application, (ii) describe the ontology and the inference methods of the system, (iii) score the definitions in the ontology on the domain-specificity and method-specificity attributes, and (iv) put the definitions in the appropriate library theories. The next sections will elaborate and illustrate each of these steps.

3.3.1. Start with an existing application

The definitions that are most likely to be usable for medical KBS development are the definitions that are already employed in existing systems. Therefore, the initial library is based on analyses of such systems. We will use CASNET (Weiss, Kulikowski, Amarel & Safir, 1984) as an example. CASNET allows the representation of causal associational networks that describe processes of diseases and has been used to build an application for diagnosing glaucoma.

CASNET was chosen as an example for several reasons. Firstly, as a shell it provides a framework for developing applications in various medical domains. Therefore it is likely that its domain ontology is a good candidate for reuse across domains. Secondly, in addition to this general causal network ontology CASNET provides idiosyncratic ontological distinctions required by CASNET's reasoning methods. This combination of properties makes CASNET an attractive illustration for developing method-specific extensions to the library. To illustrate the idea of domain-specific extensions, we have added the concept `glaucoma`, which was used in the glaucoma application developed with CASNET, to CASNET'S ontology.

3.3.2. Model the application

It is often the case that existing medical KBS do not have explicit descriptions of the underlying domain ontologies. In these cases, it is up to the library builders to define such an ontology. This is done in three steps: (i) scoring the current application on the domain-specificity and method-specificity attributes, (ii) retrieving the concepts from the library that could be useful for modelling the ontology of the application and (iii) defining the additional concepts necessary for the application's ontology.

As described in Section 3.2, the possible values of the domain-specificity and method-specificity attributes are specified in the corresponding hierarchies. CASNET is a general shell for medical applications, but since we have added the concept `glaucoma`, the system is assigned the value ''opthalmology'' on the domain-specificity attribute. For the method specificity, we concentrate on the abduction step. CASNET uses a causal network for abduction, so ''abduction by tracing causal pathways'' is selected as the value for the method-specificity attribute. Actually, CASNET uses a specialization of this method, but as yet this specialization—which we will call the CASNET method—is not represented in the method hierarchy.

When the application is scored on the method-specificity and the domain-specificity attributes, the concept definitions that are already available in the library can be retrieved. To complete the application ontology, the library builder has to define the additional classes, relations and functions required for the application. For the purpose of library construction, these newly defined concepts are the important ones. Because the method actually used by CASNET is not in the method hierarchy, the library builder also has to model the method of the system.

CASNET's application ontology. Applications build with CASNET have an explicit representation of a network, the nodes of which represent pathophysiological states. The links in the network represent causal relations between the states. States are labelled with a confirmation status, which must be one of `confirmed`, `denied`, or `uncertain`. The evidence for the confirmation status of a state comes from patient

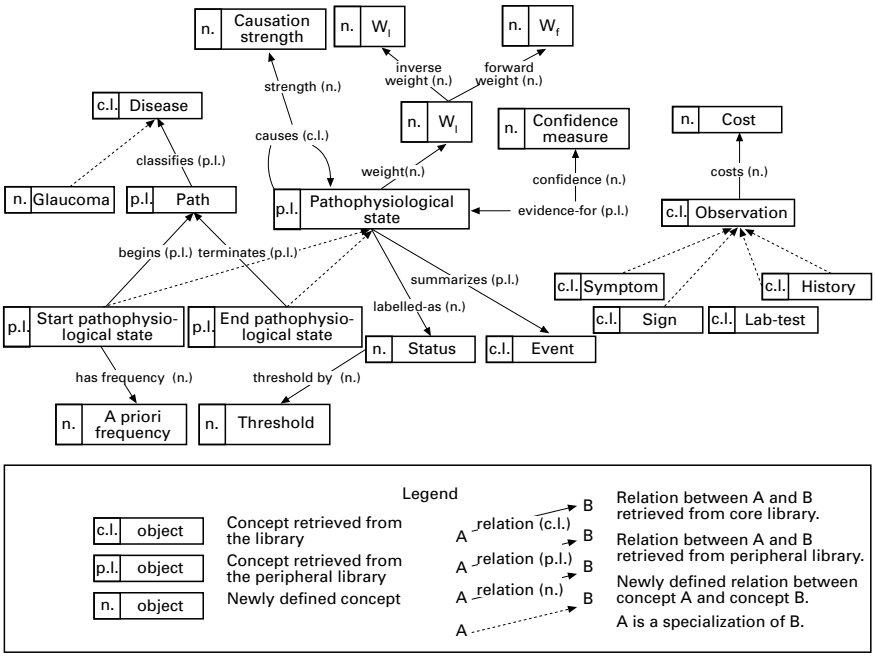

FIGURE 10. The application ontology of CASNET represented as an ontological semantic network.

observations. A specific state of the network is interpreted in terms of diseases in various states of progression.

Figure 10 presents parts of the reconstructed application ontology of CASNET in the form of an *ontological semantic network* (Abu-Hanna, 1994). There are three types of definitions in the application ontology: (i) definitions retrieved from the core library, (ii) definitions retrieved from the peripheral library and (iii) new definitions. The concepts retrieved from the peripheral library are specific for the causal tracing method, but generic across all the different specializations of causal tracing. None of the concepts that were retrieved from the opthalmology specific extensions were used for the application ontology. As mentioned above, the newly defined concept `glaucoma` was added to the application ontology as a sub-type of disease. To decide on the method specificity of the newly defined concepts, the CASNET method must be modelled for analysing the ontological requirements of the method.

CASNET's inference methods. The analysis of CASNET'S inference methods is based on the STModel. We will describe the methods for abduction and for ranking.

As mentioned, ''CASNET abduction'' is a specialization of ''abduction by tracing causal pathways between findings and disease''. The method consists of three primitive procedures. The first of these uses the evidence links between observations and states and their associated confidence measures to compute the confidence measure of the state. The second procedure then labels the states with `confirmed`, `denied` or `undetermined` by applying a threshold to the confidence measure of the states. Finally, the third procedure classifies paths of labelled states with no denied states as diseases.

The problem-solving method that CASNET uses for the ranking inference [Figure

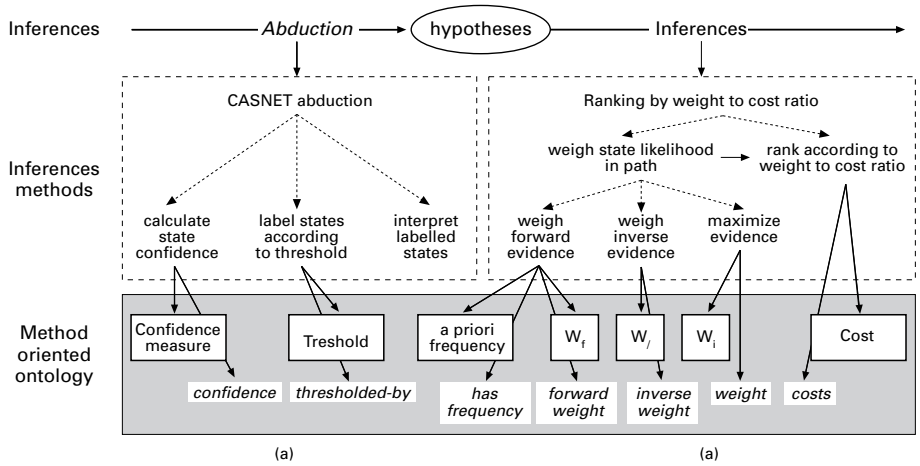

FIGURE 11. The inference methods used in CASNET for abduction of hypotheses (a) and ranking of hypotheses (b).

11(b)] consists of two procedures: weighing the evidence for the hypothesized diseases and ranking the diseases according to the weights of the evidences. The weighing procedure consists of three steps, which use the strengths of the causal relations between the states. The total weight of a state is the maximum of the forward and the inverse weights. The forward weight of a state summarizes the weight of the evidences coming from the causes of that state. The inverse weight summarizes the weight of the evidences coming from the effects of the state. When the status of a state is `undetermined`, the starting state's *a priori* frequency is used for the calculation of its forward weight. The procedure that ranks states (hypotheses) uses the ratio of the weight of the hypothesis and the costs of testing that hypothesis.

Figure 11 shows the methods that perform the abduction inference [Figure 11(a)] and the ranking inference [Figure 11(b)]. The figure also shows some of the ontological commitments that are required by the method.

3.3.3. Scoring the definitions

When the ontology of the application has been specified, the newly defined concepts must be indexed and stored in the library. In the case where the core library is largely complete, this is not difficult. The newly defined concepts are then all method or domain specific, and must be stored in the peripheral part of the library. In the case where the core library is also incomplete, the indexing is more difficult. In that case the library builder has to decide whether the definition represents a basic category of medical knowledge, or whether it is a method- or domain-specific extension. The procedure to follow in this situation is based on the principle that the concepts in the core library are intended to be reusable across many tasks and domains. If the library builder estimates that this is true for a concept under consideration, it is stored in the core library, otherwise it is considered as an extension. Of course, the subjective estimates of the library builder are not error-proof, but at present this is the only method available. One of the

assumptions that underlie this approach to library construction is that there are only a small number of truly basic categories of medical knowledge, so that it is likely that the current core library is already more or less complete.

For CASNET, scoring the definitions on the method-specificity attribute amounts to deciding whether the concepts that are newly defined in Figure 10 are all specific for the inference methods of CASNET and not for their parents in the method hierarchy. Again this requires a subjective estimate of the library builder. After inspecting how the new concepts relate to CASNET'S methods (see Figure 11) it is decided that the newly defined concepts are specific for the methods employed by CASNET, except for `glaucoma`. Because `glaucoma` does not add method-specific aspects to the definition of its super concept, the core library concept `disease`, it is assigned the value ''medical method'' on the method-specificity attribute. ''Medical method'' is the root of the method hierarchy.

The concepts must also be scored on the domain-specificity attribute. In the general case, this requires medical expertise. For the current application, deciding on the domain specificity of concepts is straightforward because CASNET was developed as a general shell for medical applications. Therefore, all the concepts—except `glaucoma`—get the value ''Medicine'' on the domain-specificity attribute, which is the root of the domain hierarchy. `glaucoma` is assigned the value ''glaucoma management''.

In summary, `glaucoma` is indexed as ''specific for the domain of glaucoma in all medical methods''. The other newly defined concepts that are used for abduction in CASNET (e.g. `confidence-measure` and `threshold`) are indexed as specific to ''CASNET abduction in medicine''. Figure 12 shows how the newly defined concepts are scored on the method-specificity attribute.

3.3.3. Storing the definitions in the library

When the definitions are scored, they must be stored in the proper parts of the library. For the new concepts with the domain-specificity value ''Medicine'' and the method-specificity value ''CASNET abduction'' or ''CASNET ranking'', two new theories are created in the library: ''CASNET abduction in medicine'' and ''CASNET ranking in medicine''. For `glaucoma`, the theory ''glaucoma management in medical methods'' is added to the library. Because ''glaucoma'' was not yet part of the domain hierarchy it is added as a specialization of opthalmology. Finally, the methods employed by CASNET must be added to the method hierarchy. For example, ''CASNET abduction'' is added as a specialization of ''abduction by tracing causal pathways between findings and diseases''.

3.4. USING THE LIBRARY

This section explains how the library can be used for constructing a part of the application ontology of a KBS. Basically, this amounts to classifying the domain of the application in terms of the domain hierarchy and specifying the methods that the application will use in terms of the method hierarchy. When a domain or a method is not in the hierarchy, the most specific ''super domain'' or ''super method'' must be used. For example, if one is building an application in the field of cardiovascular

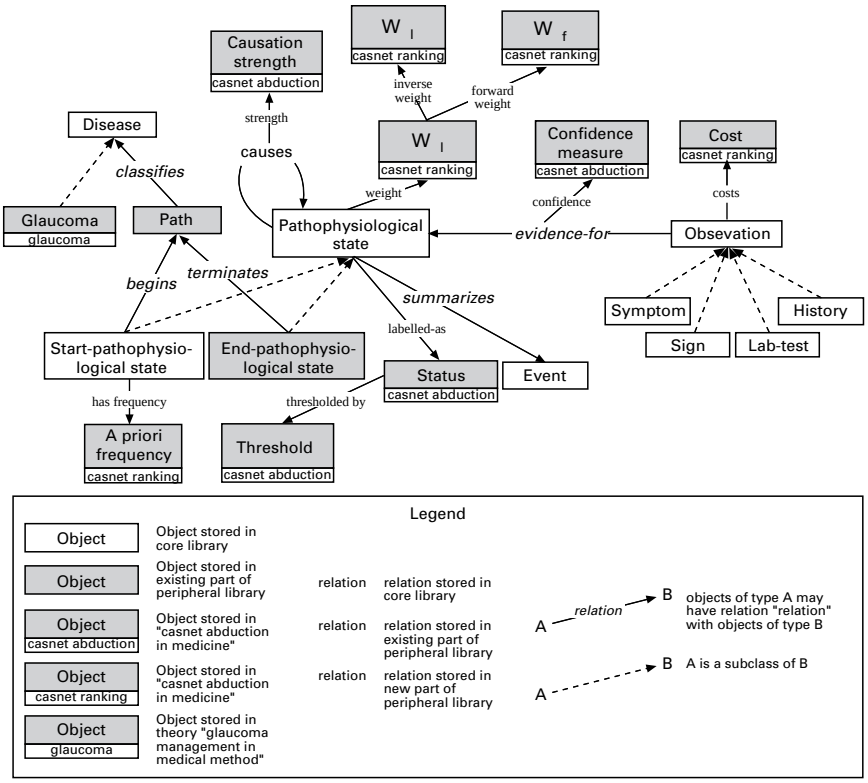

FIGURE 12. The application ontology of CASNET. The labels associated with new concepts show the method-specificity values of these concepts.

medicine and the domain hierarchy does not have an entry for this domain, internal medicine should be used instead (see Figure 7).

When the application is scored on the indexes, the library can be used to collect the concepts that are likely to be useful for the application. The peripheral theories that are included in the application ontology must satisfy two criteria. First, the theories must have a domain-specificity index that is equal to—or subsumes—the domain-specificity value of the application. Secondly, the theories must have a method-specificity index that is either equal to—or subsumes—the method-specificity values of the application. For example, for a system that uses the method ''abduction by tracing causal pathways between findings and diseases'' in the domain of cardiovascular medicine, the library would suggest including the theories P-1 and P-2, but not P-3 from Figure 9.

For retrieving definitions from the core part of the library, the indexes cannot be used. However, concepts defined in the peripheral parts of the library are often defined as specializations of core ontology concepts. In this case, the peripheral theories and the core library theories are connected by means of inclusion relations. When theories include other theories, the included theories are also automatically retrieved. In cases where core ontology theories are needed which are not retrieved because of the inclusion relations, it is up to the library user to select these theories.

For a particular application ontology and a particular reasoning step in the

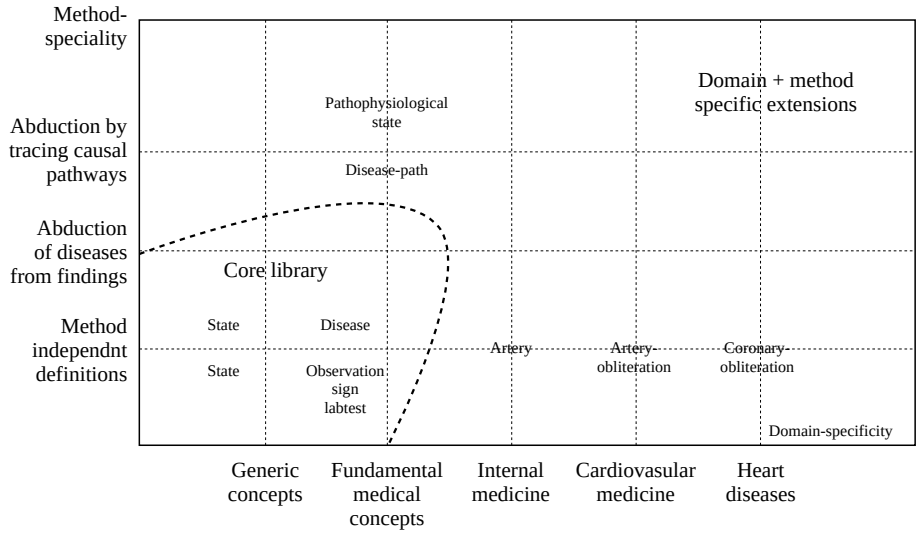

FIGURE 13. A diagram showing some definitions that are suggested for inclusion in the application ontology of a system that diagnoses heart diseases. The positions in the diagram reflect the locations of the definitions in the library.

STModel, the reusability characteristics of the definitions can be illustrated in a *reusability diagram.* Figure 13 shows such a diagram for abduction in an application that diagnoses heart diseases. The domain-specificity axis of the diagram is constructed by starting from the specific domain in the domain hierarchy, and then moving upwards through the domain hierarchy. Each of the parent nodes in the hierarchy is used as a value on the domain-specificity dimension. The method-specificity axis is constructed in a similar way, using the method hierarchy.

The region at the lower left part of the diagram contains the definitions that are retrieved from the core part of the library, which was described in Section 3.2.2. The definitions that are both method independent and generic are retrieved from the theory generic-concepts. For the other definitions in this region, the positions in the reusability diagram do not reflect from which theories they originate.

3.5. SUMMARY

The starting point of the work presented in this section is the observation that, although the potential merits of libraries of reusable ontologies are widely recognized, there are few libraries available today. Ontology libraries could provide building blocks for an application ontology, which is a specification of all the ontological distinctions that are required to perform a particular task in a particular domain. Two reasons were identified to explain the unavailability of such a library: the hugeness problem and the interaction problem.

We have presented an analysis of these problems in the context of medical knowledge, and suggests ways to make them manageable. In short, the interaction problem is addressed by the introduction of a method-specificity attribute for concepts, based on a classification of inference methods. To the hugeness problem

there are two aspects: the large number of concepts makes *building* the library a daunting exercise, and it also complicates *retrieving* the appropriate concepts when they are needed. Because of the first aspect, we have concentrated on the formulation of procedures for augmenting an initial library. The other aspect of the hugeness problem is addressed by the introduction of a domain-specificity attribute, similar to the method-specificity attribute. Based on this analysis and an analysis of the intended use of the library, three principles have been identified that can be used to impose a structure on an ontology library: organizing concepts according to (i) natural categories, (ii) inference methods, and (iii) domain division in practice. The first of these principles advocates structuring ontologies of medical knowledge according to ''topics'' that often recur in medical practice. These general categories are located in the core part of the library. The importance of this organizational principle is that it provides anchors for the more specialized concepts in the other part of the library, thereby ensuring that concepts that are defined in different ways for different methods or subdomains, have at least some common ground. The second principle says that inference methods should be used as an index for the ontological distinctions that they introduce. This facilitates the construction of application ontologies because it is easy to find out which domain concepts are required for a particular inference method. The third principle suggests that domain concepts that are specific to a particular branch in medical practice should be indexed by that sub-domain. This facilitates the construction of application ontologies because it can be used to suggest concepts that are specific for problem solving in that domain, and it also suggests what kinds of external knowledge will be available in the runtime environment of the KBS.

4. Model-based knowledge acquisition tools

One important role for ontologies is that they can be used by knowledge acquisition (KA) tools to direct the acquisition of domain knowledge. This issue is investigated in this section and the next. In this section, we present an overview of the ways in which tools can support the knowledge engineering process. In Section 5, we present a number of tools that exploit explicit ontologies to provide some of the types of support identified in this section.

4.1. KNOWLEDGE ACQUISITION AND MODELLING

One of the recurring themes in the recent knowledge acquisition literature is that knowledge acquistion is a modelling activity, as opposed to the older view of knowledge acquisition as mining. There are at least two different interpretations of this modelling process. In the first interpretation, which we will call ''KA as modelling'', modelling is viewed as a bottom-up constructive process where a structure is imposed on already elicited knowledge (e.g. Ford, Bradshaw, Adams-Webber & Agnew, 1993). In the second interpretation, called ''model-based KA'', modelling is viewed as a top-down process where an abstract model is selected or constructed which is then instantiated with application-specific knowledge (e.g.

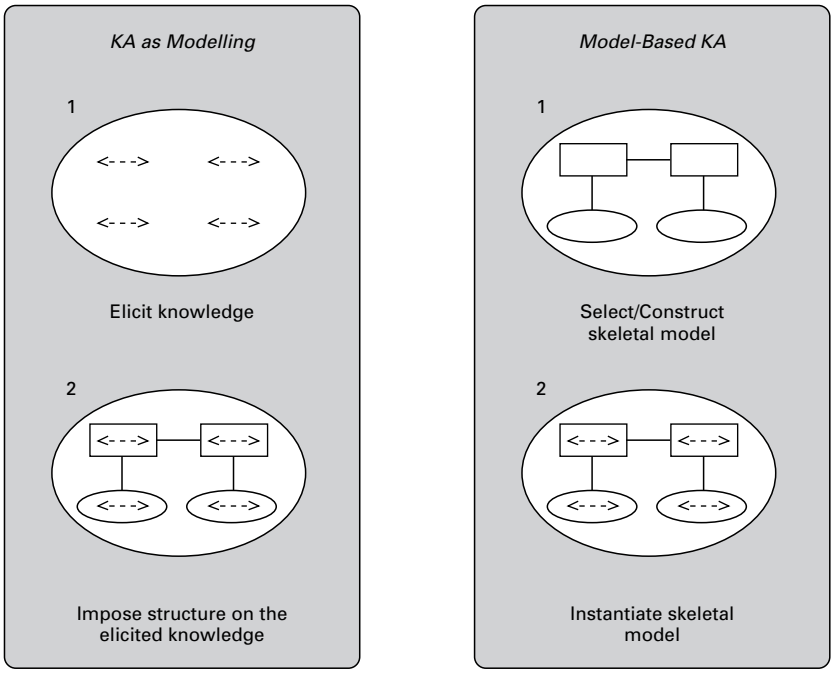

FIGURE 14. Two alternative interpretations of the modelling process in knowledge acquisition.

Breuker & Wielinga, 1989). Figure 14 shows the alternative interpretations of the modelling view.

One could argue that all forms of knowledge acquisition use some abstract model of the domain knowledge, although this model might be weak. Therefore, it is better to view the two interpretations in Figure 14 as the extremes of a continuum which ranges from weak model support to strong model support.

To gain a better understanding in the possible types of models and the way in which they can be used in the knowledge acquisition process, this section makes a comparison between a number of well-known knowledge acquisition tools. To identify dimensions on which the tools can be compared, Section 4.2 presents a general framework for describing the knowledge acquisition process according to the model-based-KA paradigm. Section 4.3 describes how this paradigm evolved. Section 4.4 then describes how each of the sub-tasks in the paradigm can be supported by tools and in Section 4.5 a number of well known tools are described and compared. Because we are mainly interested in the role of the abstract models, the comparison is biased towards tools that are closer to the model-based-KA edge of the continuum.

4.2. A FRAMEWORK FOR COMPARING TOOLS

In order to compare existing model-based KA tools, we need a general framework in which they can be described. This section proposes such a framework which distinguishes four main activities in model-based knowledge acquisition: (i) skeletal model construction, (ii) model instantiation, (iii) model compilation and (iv) model

refinement. The framework is based on an analysis of the types of support provided by existing knowledge acquisition tools.

The first of these sub-tasks, skeletal model construction, involves the creation or selection of an abstract specification of the knowledge that is required to perform a particular task in some domain. Such skeletal models may come in different flavours, and they vary in the amount of detail that they specify. For example, *generic tasks* (Chandrasekaran, 1987) specify both the method that is used to perform a task and the way that domain knowledge must be represented. In contrast, KADS *interpretation models* (Wielinga *et al.,* 1992) specify the method (using control knowledge and inference structures), but they do not specify how the domain knowledge must be represented. In the PROTÉGÉ approach (Musen, 1989*a*), both the method and the domain-specific classes are specified in the skeletal model. Here, only the instances of the classes and their relations are unspecified.

Model instantiation, the second activity in knowledge acquisition, involves ''filling'' a skeletal model with domain knowledge to generate a complete knowledge base. Many well-known knowledge elicitation tools concentrate on this activity in the knowledge acquisition process (Boose, 1985; Shaw & Gaines, 1987; Marcus, 1988; Musen *et al.,* 1988). For example, SALT concentrates on the elicitation of knowledge that is required for the Propose-and-Revise skeletal model. In the model instantiation activity the elicited knowledge is often, but not always, represented in a non-executable language.

In the model compilation activity, the instantiated skeletal model is transformed into an executable knowledge base. This subtask is only required when the instantiated model is formulated in a non-executable language.

The fourth activity in model-based knowledge acquisition is refinement of the executable model. In this activity, the dynamic characteristics of the KBS are validated using a number of selected test cases. When the KBS does not solve the test cases correctly, or produces invalid explanations, this provides feedback about erroneous or missing knowledge in the executable model. In cases where the executable model and the instantiated model are different, this activity requires ''uncompilation'': the parts of the instantiated model that correspond to the erroneous parts of the executable model must be identified.

Figure 15 shows the four basic activities in model-based knowledge acquisition. It should be emphasized that this task breakdown does not imply that the four activities are necessarily performed sequentially. As argued by Shadbolt and Wielinga (1990), the KA process is typically a cyclic process.

The dotted arrow in Figure 15 represents a fifth activity in the paradigm: the use

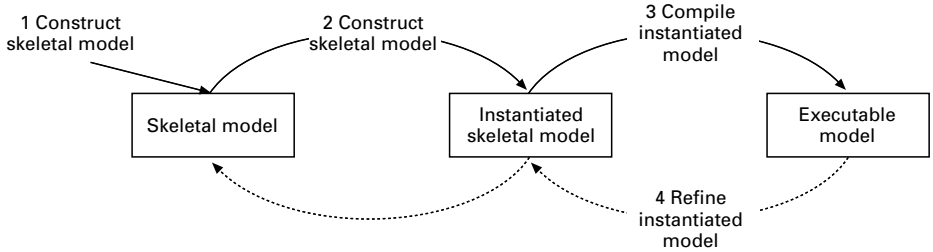

FIGURE 15. The four basic activities in model-based knowledge acquisition.

of the instantiated model to provide feedback about the validity of the skeletal model. Negative feedback can be used for identifying parts of the skeletal model that need to be adapted while positive feedback could for example be used for deciding to put a model—or parts of a model—in a library. Although, this activity is obviously important, there are at present no tools that support this activity.

The four activities require different kinds of expertise and different types of support tools. Whereas the model construction activity is inherently difficult and requires the expertise of a knowledge engineer, the knowledge instantiation activity can often be performed by domain experts, after some initial explanation of representation and tool usage. The compilation activity requires the expertise of a computer programmer, but in many existing KA tools this activity is fully automated. The knowledge refinement activity can be performed by domain experts, provided that they understand the control regime of the inference engine (Davis, 1979).

4.3. EVOLUTION OF THE PARADIGM

This section gives a historical overview of the developments in automated knowledge acquisition, illustrated with references to some well known tools that have been described in the literature. The overview is mainly intended to sketch trends in the history of knowledge acquisition tools. To emphasize these trends, the presentation is not completely chronological.

4.3.1. Ancient times: rule editors

Knowledge acquisition tools of the first generation were derived from existing expert systems. For example, KAS (Duda, Gasching & Hart, 1979), EMYCIN (van Melle, 1979) and EXPERT (Weiss & Kulikowski, 1979) were derived from PROSPECTOR, MYCIN and CASNET respectively. Tools of this era assumed that the domain expert or the knowledge engineer was able to build an initial knowledge base without extensive (tool) assistance. Only after this initial knowledge base was available could the tools support the KA process by providing feedback about the origin of erroneous solutions. The power of these tools was solely based on the explanation facilities of their inference engines, which facilitated the job of locating missing or incorrect parts of the knowledge base.

Thus, of the activities described in Section 4.2, only model refinement was extensively supported by these tools. They barely supported model construction, while the support for model instantiation was limited to symbol-level facilities such as rule editors.† Tools of this generation could not provide much support for model instantiation because of their weak skeletal models. For example, EMYCIN's skeletal model only specified that the reasoning method is backward chaining and that the domain knowledge is organized in a context tree, a simple hierarchy of domain entities. Because in this early systems the instantiated model was formulated in terms of directly executable representation formalisms no compilation activity was required.

A notable exception with respect to the lack of support for model instantiation was TEREISIAS (Davis, 1979). Like other tools of its generation, TEREISIAS did not

† Some tools of this generation used simple template-based natural language front-ends to hide the most deterrent details of the underlying representation formalisms.

have an explicit model of the types of knowledge that had to be elicited, but during the knowledge elicitation process TEREISIAS used already elicited knowledge to formulate expectations about what other knowledge might be needed. These expectations, which were represented as *rule models,* served a similar purpose as explicit skeletal models. However, because the rule models were not available at the beginning of the model instantiation phase, they could only provide support after a significant amount of knowledge had already been elicited.

Another exception is ROGET (Bennett, 1985). This system was especially designed to support the early phases of knowledge engineering. The system helps the domain expert to design the *conceptual structure* of a target consultation system, by interacting with the expert. ROGET is able to provide guidance because it has access to a library of conceptual structures of existing knowledge-based systems. These conceptual structures can be considered as kinds of skeletal models. ROGET was able to translate the constructed conceptual models into EMYCIN context trees. ROGET is very different from the other first-generation tools and could best be viewed as an early predecessor of the model construction tools of third-generation workbenches.

4.3.2. Mediaeval times: task- and method-specific architectures

Whereas the first generation of tools merely supported model refinement, the second generation of knowledge acquisition tools also supported the model instantiation activity. This higher level of support for model instantiation was a result of the fact that these tools acquired knowledge in a form that was more intuitive to the domain experts than the production rules in the earlier tools. Put differently, tools of this generation were capable of knowledge-level communication with domain experts. The type of support provided by second-generation tools can be classified into three categories.

A first reason why tools of this generation could better support model instantiation than their ancestors was because of their more restrictive skeletal models. For example, tools as MOLE (Eshelman, 1988) and SALT (Marcus & McDermott, 1989) used knowledge of problem-solving methods such as Cover-and-Differentiate and Propose-and-Revise to engage in a structured dialogue with the domain expert. Because these tools knew what kind of knowledge was required for these methods, they could strongly focus the knowledge elicitation dialogue. Another system in this category was OPAL (Musen *et al.,* 1988). OPAL did not only make assumptions about the method that the KBS was going to perform, but also about the domain that the system would reason about: oncology. Because of this, OPAL was able to communicate with domain experts in domain-specific terminology.

A second way in which tools of this era bridged the gaps between the ways in which humans process knowledge and the ways knowledge is represented in AI formalisms was the use of graphical user-interfaces. For instance, the graphical user-interface of OPAL allowed the oncologists to enter knowledge in forms that resembled the paper forms that they were used to working with. The tool then automatically translated these forms into an internal representation which was subsequently compiled into production rules.

A third group of tools of this generation based their support on interviewing techniques which originated from psychology. Typical examples of this category are tools such as ETS (Boose, 1985) and its successor AQUINAS (Boose & Bradshaw,

1988), which embody the repertory grid technique and ALTO (Major & Reichgelt, 1990), which is based on the laddering technique.

With respect to the activities described in Section 4.2, tools of this generation provided stronger support for the model instantiation activity than their predecessors because they used stronger skeletal models. However, these skeletal models were hardwired in the tools: it was not possible to edit skeletal models or construct new ones. Thus, the model construction activity was not supported. Some of these tools acquired the knowledge in a format that was not directly executable. For these tools, an explicit compilation activity was required. This compilation was usually automatic and could not be controlled by the knowledge engineer. Although some of the tools of this generation had a refinement facility similar to those of the earlier tools, the emphasis was on getting the knowledge model right the first time.

4.3.3. Modern times: integrated KA workbenches

Only recently tools have been built that support skeletal model construction. In contrast with the systems of the first and the second generation, which are often presented as stand-alone programs, these tools are usually embedded in larger knowledge engineering environments, called KA workbenches. One of the first systems that supported model construction was PROTÉGÉ (Musen, 1989*a*). This tool uses an abstract model of a problem-solving method, and allows the knowledge engineer to associate the knowledge roles of the method with domain-specific labels. Based on these associations, PROTÉGÉ can be used to generate model instantiation tools such as OPAL, which interact with experts in domain-specific terminology. A limitation of PROTÉGÉ is that it is based on one problem-solving method: episodic skeletal-plan refinement. Other systems of this generation allow the construction of arbitrary skeletal models from sets of primitive components.

Most of the existing KA workbenches concentrate on the earlier activities of the model-based knowledge acquisition paradigm. Typically, skeletal-model construction is supported by libraries of model components which can be selected and adapted for an application by means of specialized editors. The model instantiation activity is supported by tools that exploit the explicitly represented skeletal model to focus the elicitation activity. In most workbenches, the instantiated skeletal model is formulated in a non-executable language, so compilation is required. This compilation may or may not be automatic. In most of the workbenches there is little emphasis on the knowledge refinement activity.

4.4. TYPES OF TOOL SUPPORT

In Section 4.2 a framework was presented for comparing model-based knowledge acquisition tools which distinguishes four activities. This section identifies for each of these activities ways in which they can be supported or automated by knowledge acquisition tools. Section 4.5 will use the results of this analysis to make a classification of a number of well-known knowledge acquisition tools.

4.4.1. Supporting skeletal-model construction

As mentioned, different types of skeletal models have been proposed in the literature. Here we will adopt the view that skeletal models consist of a task model and an application ontology (this is also the view in CommonKADS and PROTÉGÉ-II). Together, these specify what kind of application knowledge is required to solve

problems in the application domain. The most obvious form of support for the construction of skeletal models is by means of (graphical) editing facilities. Dependent on the expressiveness of the formalism in which the models are expressed, such editors can perform several types of consistency checking and completeness checking. A second type of support is by providing libraries of primitive components and typical configurations of those. Usually, these two types of support are combined: libraries provide generic configurations that can be fine-tuned for particular applications using specialized editors. The third, most ambitious type of support for skeletal model construction is process support. This type of support requires a prescriptive theory of how skeletal models should be constructed from primitive components. For task models, such a theory is currently under development (e.g. Aben, 1995). However, for application ontologies such a theory is not yet available. Section 3 will describe principles that can be viewed as a first sketch of such a theory.

4.4.2. Supporting model instantiation

In general, stronger skeletal models allow better support for model instantiation because it can more easily be determined which knowledge is valid and required for problem solving. There are five ways in which model instantiation can be supported: (i) checking if the entered knowledge is consistent with the skeletal model, (ii) checking whether the entered knowledge is all the knowledge that is required according to the skeletal model, (iii) using domain specific terminology, (iv) using intuitive visualization techniques, and (v) use of a structured dialogue.

The simplest form of support, consistency checking (e.g. syntax checking, type checking), can also be found in conventional programming tools such as syntax-driven editors. However, the other types of support are dependent on the stronger skeletal models used in knowledge engineering. For example, there are two types of completeness checking: checking whether for all the knowledge types knowledge has been elicited and checking whether all the knowledge of a particular type has been elicited. To provide this support, the skeletal model should define which knowledge types there are in the domain and what the constraints are on the quantity of the knowledge pieces for each of these knowledge types. The use of domain-specific terminology requires that this terminology is defined. Also, specialized visualization techniques are based on strong assumptions about what is to be visualized. For example, when it is assumed that the knowledge in the domain consists of objects and values of these objects on a number of dimensions, it can be decided to use grids for visualizing the knowledge. Dialogue structuring requires—amongst other things—the ability to find out what knowledge still needs to be elicited and is therefore dependent on the capacity to perform completeness checking, which in turn requires a strong skeletal model.

4.4.3. Supporting model compilation

The model compilation activity is only required when the knowledge is elicited using a non-executable language. The advantage of such knowledge-level languages is that they facilitate model-instantiation because they are easier to understand for non-programmers. Therefore, it is to be expected that for tools that give better support for model instantiation the compilation activity is more complex.

In some tools, the compilation activity is completely automated. Once the skeletal model is instantiated, the tool is able to generate an executable knowledge base without further assistance. This type of support for compilation may seem the most powerful type, but there is a serious drawback. Compilation becomes more difficult when the source language is more expressive. With current compilation technology, automatic compilation would in many cases yield computationally inefficient executable models.

Alternatively, tools may engage in an interactive compilation dialogue. Here the tool attempts to compile the instantiated model automatically, but whenever the compiler does not have sufficient information to select an appropriate representation, it may ask for additional information to resolve the ambiguities.

In a third group of tools compilation is considered as an activity to be performed by the knowledge engineer. Here, the emphasis is on supporting the knowledge engineer instead of on replacing the knowledge engineer. Manual compilation can be supported by providing libraries of reusable compilation procedures from which the knowledge engineer can select the most appropriate one.

4.4.4. Supporting model refinement

Model refinement, which takes place in the context of a running system, may be supported in four ways. Firstly, the tool may provide a tracing facility that shows the reasoning steps that lead to the solution that the system arrived at. This type of support is also provided by tools that support software engineering. Secondly, tools may be able to inspect such traces to answer *why* and *how* questions. For such a facility, the tool must have a persistent representation of the trace. The explanations of first-generation tools were for example based on such persistent traces. A third type of support is the ability to answer *why not* and *what if* questions. Such a facility requires that the tool is capable of hypothetical reasoning. Finally, tools may be able to locate the missing or incorrect knowledge pieces in the knowledge base that are responsible for the erroneous solution. That is, they are capable of blame assignment.

4.5. KNOWLEDGE ACQUISITION TOOLS

In this section, a number of existing knowledge acquisition tools are analyzed with respect to the types of support that were identified in the previous section. The tools that are described were selected because they are prototypical representatives of different classes of tools. The results of the analysis are shown in Figure 16, Figure 17 and Figure 18. For some tools, describing the functionality in terms of the four sub-tasks of the model-based KA paradigm required some reinterpretation. However, in order to be able to compare different tools it is necessary to have a common framework, and we believe that the framework presented in Section 4.2 is sufficiently general to do justice to the particularities of the different tools. The tools are described in a roughly chronological order.

4.5.1. Emycin

EMYCIN (van Melle, 1979), a shell based on the domain-independent core of MYCIN, is intended as a tool for the development of consultation programs. As already mentioned in Section 4.3, EMYCIN has a fixed skeletal model based on the

backward-chaining method and the context tree. Because this skeletal model is under-constrained, the tool can provide only limited support for model instantiation. The knowledge is entered into the system using the ''Abbreviated Rule Language'' a user-friendly interface on top of the production rules that EMYCIN uses for representing knowledge. A rule that is entered is checked for syntactic validity, and to a limited extent for ''semantic'' validity: EMYCIN checks whether an entered rule does not directly contradict another rule, and whether the entered rule is subsumed by another rule. Because the entered knowledge is directly mapped onto production rules there is no need for compilation. To support model refinement, EMYCIN has tracing facilities and it is able to answer why and how questions.

4.5.2. Kas

KAS (Duda *et al.,* 1979), which is derived from PROSPECTOR, is very similar to EMYCIN but has richer facilities for supporting model instantiation. For example, the tool protects against errors such as disconnecting parts of the semantic network that KAS uses for knowledge representation. Further, the tool keeps a record of unfinished elicitation jobs, thereby performing a kind of completeness checking.

4.5.3. Expert

Compared to the previous tools, EXPERT (Weiss & Kulikowski, 1979) makes stronger assumptions about the kinds of knowledge that must be elicited: findings and hypotheses. It distinguishes three kinds of rules: finding-to-finding rules, hypothesis-to-hypothesis rules and finding-to-hypothesis rules. However, although EXPERT has a stronger skeletal model than EMYCIN and KAS, the tool does not use this model for supporting model instantiation. In EXPERT, the rules need to be entered using text editors. After syntax checking, these rules are then automatically compiled into an efficient internal representation. For refinement, EXPERT provides similar facilities as EMYCIN and KAS.

4.5.4. Mole

MOLE (Eshelman, 1988) is a knowledge acquisition tool for systems that use the Cover-and-Differentiate problem-solving method. The (built-in) skeletal model of MOLE is derived from the knowledge requirements for this method. MOLE uses its skeletal model to engage in a focused dialogue with the domain expert. During the model instantiation phase, MOLE uses static analysis techniques to decide on the consistency and completeness of the entered knowledge. Internally, the entered knowledge is represented in the form of production rules, so compilation is not needed. MOLE has advanced facilities for supporting the model-refinement activity. If MOLE makes an incorrect diagnosis in the model refinement phase, the tool tries to locate the source of the error and recommends possible remedies. Thus, in addition to tracing and explanation, this tool is also capable of blame assignment.

4.5.5. Salt

SALT (Marcus & McDermott, 1989) can be used to develop expert systems that use the Propose-and-Revise problem-solving strategy. The tool has built-in expectations about the knowledge requirements of this method to structure the knowledge acquisition dialogue. For Propose-and-Revise, there are three types of knowledge:

propose knowledge, constraint knowledge and knowledge for fixing constraint violations. SALT is capable of consistency checking and completeness checking. Initially, the entered knowledge is represented in a dependency network which is then automatically compiled into production rules. To support knowledge refinement, SALT is able to answer how, why, what-if and why-not questions.

4.5.6. Opal

As already indicated in Section 4.3, the skeletal model of OPAL (Musen *et al.,* 1988) is based on the method that the knowledge-based system will use (episodic skeletal-plan refinement) and on the domain about which it will reason (oncology). Because of this strong model, the tool is able to perform extensive consistency and completeness checks, to communicate with experts in domain-specific terminology, and to use specialized visualization techniques. OPAL compiles the entered knowledge automatically into production rules. The tool has no knowledge refinement facilities.

4.5.7. Aquinas

AQUINAS (Boose & Bradshaw, 1988) is a system for building classification expert systems. Like its predecessor, ETS (Boose, 1985), the tool is centred around the repertory grid technique, which is a psychological technique for eliciting concepts and ''personal constructs''. These personal constructs are dimensions on which the concepts may have values. The elicited concepts and distinctions together form a skeletal model that is used to direct the further KA process, which consists of assigning values to the concepts on the dimensions. In the knowledge instantiation phase, the concepts are organized in hierarchies and assigned values on the dimensions. This process is supported by graphical visualization and by dialogue structuring. The tool is able to perform simple completeness checks. The elicited knowledge can be compiled automatically into various expert system shells (e.g. KEE, EMYCIN, OPS5, etc.). The facilities for supporting knowledge refinement are those of the shells into which the knowledge is compiled.

4.5.8. Alto

ALTO (Major & Reichgelt, 1990) is a tool that implements the laddered-grid knowledge-elicitation technique, intended for hierarchy elicitation. The tool expects that knowledge can be modelled in terms of tree structures. Based on this assumption, the tool visualizes the elicited knowledge in terms of directed graphs, thereby providing the user with an intuitive overview of the elicited knowledge. During model instantiation, the tool performs consistency checking and some forms of completeness checking. For example, it insists that sibling concepts have at least one discriminating attribute. The elicited knowledge is semi-automatically translated into CommonSLOOP, an object-oriented representation system. During compilation, ALTO asks for additional knowledge for resolving ambiguities. ALTO has no facilities for supporting knowledge refinement.

4.5.9. Protégé

PROTÉGÉ (Musen, 1989*b*), which was described briefly in Section 4.3, was probably the first tool that recognized skeletal model construction as a distinct activity. The tool has a fixed task model, but allows the knowledge engineer to specify the ontology. This specification is then used to generate tools similar to OPAL, but for different application domains.

4.5.10. Spark-burn-firefighter

Another tool that supports model construction is SPARK, which belongs to the SPARK-BURN-FIREFIGHTER (SBF) environment (Klinker, Dhola, Dallemagne, Marques & McDermott, 1991). SPARK allows the user to indicate which of the tasks in a particular industrial environment could be performed by a KBS. To do this, the tool employs a general model of the tasks that are performed in some domain. Based on this analysis and a theory of what methods are used in particular organizational environments, SPARK generates a specification of the method of the KBS, in the form of a configuration of mechanisms. This mechanism configuration is then used by BURN, the model instantiation tool of the SBF workbench, to elicit the domain knowledge that is required by the method. For each mechanism in the library, BURN has a specific knowledge acquisition module. It is not entirely clear from the literature which kinds of support are provided by BURN. As explained in (Yost, Klinker, Linster, Marques & McDermott, 1994), the knowledge refinement tool FIREFIGHTER was never implemented.

4.5.11. Keats

The KEATS system (Motta, Rajan, Domingue & Eisenstadt, 1990) consists of a number of tools that support the various activities in model-based KA. The skeletal models in KEATS are *coding sheets,* templates that define the structure of the skeletal model. In Motta *et al.* (1990) both task-oriented coding schemes and domain-oriented coding schemes are mentioned. However, it is not clear from the literature whether the system supports the development of such coding schemes. The coding schemes can be filled-in to arrive at an instantiated skeletal model. The instantiated model is then used to develop an executable knowledge base. For the executable model, KEATS uses a hybrid representation language for which it provides a forward- and backward-chaining rule interpreter, a nonmonotonic truth maintenance system, a frame-based representation language and a constraint-based language. KEATS has a number of facilities for supporting model refinement. For example, the tool has graphical visualization tools for inspecting persistent traces of problem-solving sessions at different levels of abstraction.

4.5.12. Shelley

SHELLEY (Anjewierden *et al.,* 1992b) is a workbench that provides tool support for the KADS methodology. The system supports model construction by providing an inference structure editor that has access to a library of interpretation models. Model instantiation is supported by a number of facilities (e.g. a concept editor, a card sort tool, a protocol editor, etc.). SHELLEY concentrates on the earlier activities of the knowledge acquisition process, and it does not produce executable knowledge bases. Therefore, model compilation and model refinement are not supported.

4.5.13. Kadstool

KADSTOOL (Albert & Jacques, 1993) is a commercial system based on the same ideas that underlie SHELLEY, but it is based on a more recent version of the KADS methodology. Besides an inference structure editor, KADSTOOL provides facilities for defining domain ontologies. The system has an editor that supports domain modelling. Some actions that this editor supports can be considered as ontology construction. For example, it is possible to define that domain relations are transitive. This knowledge can then be used to decide how domain knowledge must be visualized. However, in KADSTOOL ontology construction and model instantiation are not clearly separated. Similar to SHELLEY, KADSTOOL is intended as a tool for knowledge analysis; it does not support model compilation or model refinement.

4.5.14. Kew

KEW (Shadbolt & Wielinga, 1990; Anjewierden, Shadbolt & Wielinga, 1992*a*), another third-generation KA environment, is a large system that embodies a variety of knowledge elicitation and knowledge refinement tools. As in its predecessor SHELLEY, the skeletal models that are used in KEW are KADS-based. However, in contrast with the original KADS approach, KEW does not merely provide a library of such skeletal models, but it uses the theory of ''generalized directive models'' (GDMs) for providing active support for the model construction activity (van Heijst, Terpstra, Wielinga & Shadbolt, 1992). As in SBF and SHELLEY, the skeletal models in KEW are task models—there is no explicit notion of ontology. KEW has a number of knowledge instantiation tools, which represent the elicited knowledge in private representation languages. These private representations can be translated into a frame language and into first-order predicate logic. For both languages KEW has interpreters and knowledge refinement facilities.

4.5.15. Krest

The KREST workbench (Steels, 1993) is yet another example of a third-generation KA environment. In this system, which is based on the componential framework (Steels, 1990), skeletal-model construction consists of two parts: (i) the construction of a task structure, which is a task decomposition tree, and (ii) the construction of a model dependency diagram, which is a specification of the domain models (or ontology) that are needed to perform a particular task. For both constituents KREST provides editors. KREST is intended to be used in combination with an *application kit* which is a library of reusable components. In parallel with the task structure for the application, a task structure for knowledge acquisition is constructed. Thus, for every problem-solving method, an application kit must also have an associated knowledge acquisition method or tool. It is not clear from the literature in which ways model instantiation is supported. The elicited knowledge is compiled into CLOS code. KREST does not support knowledge refinement.

4.5.16. Dids

In the DIDS system (Runkel & Birmingham, 1994), the emphasis is on separating task knowledge and search control knowledge. The skeletal model is formed by a description of a problem space, a set of operators (the task model) and a set of knowledge structures (the ontology). Based on this skeletal model, a number of

mechanisms for knowledge acquisition are selected, and a knowledge acquisition method is specified. The mechanisms for knowledge acquisition use specialized visualizations and may have private validation procedures. In the model-compilation phase, the knowledge engineer associates problem-solving mechanisms with the operators. The mechanisms can be associated with code for compiling to different problem solvers. Because the compilation procedures make assumptions only about the mechanisms, and not about the instantiated knowledge, they can be stored for reuse once they are developed.

4.5.17. Protégé-II

Current work on PROTÉGÉ-II (Puerta, Egar, Tu & Musen, 1992) attempts to overcome the limitations of PROTÉGÉ. PROTÉGÉ-II is a large environment that embeds a number of tools, amongst which a *mechanism* configuration facility and an ontology editor. Mechanisms are primitive building blocks for problem-solving methods. In the PROTÉGÉ-II framework, the configuration of mechanisms and the application ontology together form the skeletal model. Another tool of the workbench, DASH, uses the ontology to generate a model instantiation tool, which is capable of consistency checking and communication in domain-specific terminology. The user of DASH (usually a knowledge engineer) is responsible for defining an intuitive user-interface and, to some extent, a sensible dialogue structure. Similar to PROTÉGÉ, PROTÉGÉ-II compiles the generated knowledge directly into CLIPS production rules. The system does not support knowledge refinement.

The results of the analysis of the different tools are summarized in Figure 16, Figure 17 and Figure 18. It should be emphasized that tools which have many pluses in the tables are not necessarily the ideal tools for knowledge acquisition. Some tools support all activities to some extent, while other tools provide extensive support for one activity only. For example, although AQUINAS does provide editor support for ontology specification according to Figure 16, the type of things that can be specified are restricted to concepts and dimensions. Further, there is always some form of subjectivity in analyses as the one presented here. This subjectivity is manifest in the tools that were selected, the dimensions that were used for the comparison and the way in which the consulted literature was interpreted. However, in spite of these difficulties we still think that the tables are a useful starting point for comparing tools and investigating which facilities an ideal knowledge acquisition tool should provide.

4.6. SUMMARY

This section has presented a framework for comparing model-based knowledge acquisition tools. The framework distinguishes four activities in the model-based KA paradigm: the construction of a skeletal model, the instantiation of that model, the compilation of the instantiated model into an executable model and the dynamic evaluation of the executable model to provide feedback about the validity of the instantiated model. Also a fifth activity was mentioned: the use of the instantiated model to provide feedback about the validity of the skeletal model, but since there are at present no KA tools that support this activity it was left out of the

	Skeletal model construction					
	Task model			Ontology		
Tool	Editor	Library	Process	Editor	Library	Process
EMYCIN	−	−	−	o	−	−
KAS	−	−	−	−	−	−
EXPERT	−	−	−	−	−	−
MOLE	−	−	−	−	−	−
SALT	−	−	−	−	−	−
OPAL	−	−	−	−	−	−
AQUINAS	−	−	−	o	−	−
ALTO	−	−	−	o	−	−
PROTÉGÉ	−	−	−	+	−	−
SBF	+	+	+	−	−	−
KEATS	?	+	−	?	+	−
SHELLEY	+	+	−	−	−	−
KADSTOOL	+	+	−	+	−	−
KEW	+	+	+	−	−	−
KREST	+	+	−	+	+	−
DIDS	?	+	−	?	+	−
PROTÉGÉ-II	+	+	−	+	+	−

FIGURE 16. A summary of the means by which the tools described in this section support the model construction activity. Editor support means that the tool has editing facilities for specifying the skeletal model, library support means that the tool provides access to a library of reusable skeletal models or components of skeletal models, and process support means that the tool actively supports the modelling process. +: the type of support is provided; o: the type of support is only provided to a limited extent; -: the type of support is not provided; n.a.: not applicable; ?: could not be determined from the literature whether the type of support is provided.

framework. For each of the four activities, ways were distinguished in which they could be supported by tools. Then, a number of well-known KA tools were compared with respect to these types of support.

An important reason for developing the framework was to gain insight in the range of skeletal models that have been used in KA tools and the effects of the use of these models on the other knowledge acquisition activities. Although it was claimed that all knowledge acquisition tools are model based to some extent, the model-based-KA perspective has introduced some bias in the selection of dimensions for comparison. For example, a number of researchers in the KA-as-modelling paradigm have emphasized the importance of using multiple experts (e.g. Shaw & Gaines, 1989). Tools developed from this perspective often have advanced features for integrating the opinions of different experts, but this feature was not chosen as a dimension for comparison.

Based on the analysis, the following conclusions can be drawn. Firstly, different types of skeletal models have been used in model-based knowledge acquisition, which contain different types and different amounts of information. The skeletal models can be classified according to a number of dimensions. There are

- tools which use widely applicable but weak skeletal models (e.g. EMYCIN) and tools which use very a specific skeletal model which have a limited scope (e.g. MOLE);

	Model instantiation				
Tool	Consistency checking	Completeness checking	Domain terminology	Intuitive visualization	Dialogue structuring
EMYCIN	o	−	−	−	−
KAS	o	o	−	−	−
EXPERT	o	−	−	−	−
MOLE	+	+	−	−	+
SALT	+	+	−	−	+
OPAL	+	+	+	+	−
AQUINAS	+	+	+	+	o
ALTO	+	+	−	+	−
PROTÉGÉ	+	+	+	+	−
SBF	+	−	−	+	+
KEATS	+	?	+	+	−
ICONKAT	+	−	−	+	−
SHELLEY	+	−	−	+	−
KADSTOOL	+	+	−	+	−
KEW	+	+	−	+	−
KREST	+	?	?	?	?
DIDS	+	+	−	+	+
PROTÉGÉ-II	+	?	+	+	−

FIGURE 17. A summary of the means by which the tools support model instantiation. Key as in Figure 16.

	Model compilation			Model refinement			
Tool	Automatic	Semi-automatic	Library support	Tracing	How/why	What if/why not	Blame-assignment
EMYCIN	n.a.	n.a.	n.a.	+	+	−	−
KAS	n.a.	n.a.	n.a.	+	+	−	−
EXPERT	+	−	−	+	+	−	−
MOLE	n.a.	n.a.	n.a.	+	+	?	+
SALT	+	−	−	+	+	+	−
OPAL	+	−	−	+	?	?	−
AQUINAS	+	−	−	n.a.	n.a.	n.a.	n.a.
ALTO	−	+	−	+	−	−	−
PROTÉGÉ	+	−	−	?	?	?	−
SBF	+	−	−	+	−	−	−
KEATS	?	?	?	+	+	−	−
ICONKAT	+	−	−	+	+	n.a.	n.a.
SHELLEY	−	−	−	−	−	−	−
KADSTOOL	−	−	−	−	−	−	−
KEW	o	−	−	+	+	−	−
KREST	+	−	−	+	−	−	−
DIDS	−	−	+	+	−	−	−
PROTÉGÉ-II	+	−	−	+	−	−	−

FIGURE 18. A summary of the means by which the tools support model compilation and refinement. Key as in Figure 16.

- tools which use skeletal models of a knowledge representation formalism (e.g. EXPERT) and tools which use knowledge-level skeletal models (e.g. SHELLEY);
- tools which use fixed, implicit skeletal models (e.g. OPAL) and tools which allow user-defined, explicit skeletal models (e.g. PROTÉGÉ-II);
- tools where the skeletal model is a model of the reasoning processes (e.g. KEW) and tools where the skeletal model is elicitation oriented (e.g. AQUINAS); and
- tools where the skeletal model describes the types of domain knowledge (e.g. ALTO) and tools where the model describes the knowledge requirements of the problem-solving method (e.g. SALT).

In model-based knowledge acquisition, the information in the skeletal model is used to support model instantiation, model compilation and model refinement. Therefore, it is important to know how the different types of information that skeletal models may contain are related to the types of support that can be provided for the other activities. In the next section, this question is addressed for the model instantiation activity. The section describes the CUE workbench which uses information specified in one part of the skeletal model—the application ontology—to support model instantiation by consistency checking, by completeness checking, by using domain-specific terminology, by using specialized visualization and by dialogue structuring.

Section 6 addresses the question of how skeletal models can be used in the model compilation activity. It presents an approach where user-defined, knowledge-level skeletal models are mapped onto skeletal models of the representation formalisms used by problem solvers. These mappings may be considered as extensions to the knowledge-level skeletal model for compilation purposes.

5. Ontology-based knowledge acquisition in CUE

The analysis in the previous section of the state of the art with respect to tools that support the model-based knowledge acquisition paradigm shows that there is an emerging theory of the various activities in this process and of the ways in which tools can support these activities. We now present CUE as a KA environment that operationalizes this theory. Many of the ideas behind CUE are thus not new, but are just an explicit integration of principles that underly existing tools. CUE was developed as a testbed for extending this emerging KA process theory, in particular in the areas of (i) exploiting a *library of ontologies* such as the one described in Section 3, and (ii) the exploration of the notion of *knowledge-elicitation strategies.* The library of ontologies acts as a repository of previous knowledge engineering experiences, and enables the knowledge engineer to reuse descriptions of the structure of domain knowledge that have proven useful in the past. Knowledge-elicitation strategies are principles for organizing the knowledge-elicitation dialogue, and present an alternative for existing techniques that either have predefined dialogue structures or leave navigation to the user.

As mentioned in Section 2 we distinguish four activities in knowledge modelling: building a task model, building an application ontology, mapping the task model

TABLE 2
The activities in constructing the knowledge model that are distinguished in Section 2 approach, and the CUE *tools that support these activities*

Modelling activity	CUE tool
Construct task model for application	QUITE
Construct application ontology	QUOTE
Map task model onto application ontology	QUITE
Instantiate application ontology	QUAKE

onto the application ontology and instantiating the application ontology (see Figure 4). Table 2 shows the CUE tools that support these activities.

Section 5.1 presents a global overview of how the CUE tools are intended to be used in the KA process. Section 5.2 describes the two tools that support skeletal model construction in CUE: QUITE, a task modelling editor and QUOTE, an editor for ontologies in Ontolingua. Section 5.3 describes QUAKE, a tool that exploits the skeletal models developed with QUITE and QUOTE to elicit domain knowledge in a focused way. In the context of QUAKE an analysis of possible knowledge-elicitation strategies is presented. In Section 5.4, CUE is compared with KEW, PROTÉGÉ-II and DIDS, and some strengths and weaknesses of the system are described. This section only describes CUE's facilities for supporting model construction and model instantiation. Section 6 describes how CUE could support model compilation. The current version of CUE does not support model refinement, the fourth activity of the model based knowledge acquisition paradigm.

5.1. STEPS IN KNOWLEDGE MODELLING

To help understand why the different tools have their specific functionalities, this section presents an analysis of the different steps in the knowledge modelling process in the form of a generic scenario. In principle, each of the steps should be supported by a knowledge engineering workbench. The scenario is based on the activities mentioned in Table 2, but some activities are divided into multiple steps because different support facilities are needed.

(1) **Informally describe domain and task of the application.** KBS development always starts with getting an initial picture of the kind of application that is required. This typically requires talking to managers and domain experts, reading some publications about the field, etc.
(2) **Identify generic tasks.** Based on the informal domain and task description, the knowledge engineer then constructs an initial version of a task model. This model—a configuration of generic-task instances—is underspecified: it only models the high-level structure of the reasoning task that the application should perform. Because of the strong assumptions about the *structure* of generic tasks (the STModel), the task model still gives guidance about the types of domain knowledge that are needed.
(3) **Specify which parts of the task must be automated.** Usually, only some parts of

the reasoning process will be performed by a KBS. This should be specified in the task model.
(4) **Construct the application ontology.** For constructing the application ontology the ontology library is provided. As described in Section 3, the library is indexed by domain and by methods. By specifying the domain and the method, the library can be used for retrieving concept definitions that are likely to be useful in the application ontology. If there is no entry for the domain in the library, constructing the application ontology requires more creativity from the knowledge engineer.†
(5) **Specify the role-to-role mappings.** When the application ontology has been built, the task model can be completed. This involves specifying which knowledge roles are shared between the generic-task instances, by means of role-to-role mappings.
(6) **Map task model onto ontology.** The connection between the task model and the application ontology must be specified. This can be done by defining ontology mappings. These mappings specify that particular roles in the task model may only be fulfilled by instances of particular ontological concepts. Once the mappings have been specified, the skeletal model is completed.
(7) **Create elicitation agenda.** Now the application ontology must be instantiated. The first step in this process is creating an elicitation agenda—a list of elicitation activities that should be performed.
(8) **Specify knowledge-elicitation strategy.** After having defined the elicitation agenda, a knowledge-elicitation strategy may be specified. This can best be viewed as an ordering on the elicitation activities in the agenda.
(9) **Elicit domain knowledge.** Finally, the tuples and instances of the relations, functions, and classes in the application ontology should be elicited and saved in a knowledge repository.

In principle, all of the steps in the generic scenario should be supported by CUE tools. In the following sections, it will be explained to what extent this is realized in the current CUE implementation.

5.2. SKELETAL MODEL CONSTRUCTION IN CUE

As mentioned in Section 2, in our approach the skeletal models consist of a task model and an application ontology. For both components, CUE contains a tool that supports their construction. QUITE, the task model editor, graphically supports the configuration of STModel instances into a task model for the application. QUOTE supports construction of application ontologies.

5.2.1. Quite

In Section 4 it was argued that there are three ways in which tools can support the construction of task models: (i) by providing specialized editors, (ii) by providing libraries of reusable components, and (iii) by providing support for the modelling process.

In the context GAMES-II, library support is quite easy. Because the task models are configurations of the STModels for diagnosis, therapy planning and patient monitoring, the task modelling library contains only three models. QUITE users can

† Section 7 will present a number of guidelines for acting in this situation.

select instances of these STModels and connect them by means of *control links* and *role-to-role mapping*. Control links are global indications of the order in which the instantiated STModels must be invoked to perform the application task. They are intended for supporting knowledge acquisition and do not contain sufficient information to drive problem solving. The nitty gritty of task control is specified in the design model. Role-to-role mappings indicate data-flow between STModel instances. For example, they could specify that diagnostic hypotheses are mapped onto therapeutic problems.

QUITE users can specify control links and role-to-role mappings by setting the tool in the appropriate mode and drawing lines between the objects that need to be connected. If the specified links or mappings do not violate syntactical constraints, they are added to the task model. An example of a syntactical constraint is that roles should not be mapped onto roles within the same instance of a generic task.

In many cases, the KBS that is being developed will only automate parts of the problem solving process, while other parts remain the responsibility of human agents. QUITE supports this by allowing the user to indicate which of the inferences in the task model will be performed by the application. The other tools in CUE will only attempt to acquire the knowledge required for these parts of the reasoning process.

The task model is an informal, high-level overview of the medical reasoning process which fulfils two functions in the remainder of the knowledge acquisition process; (i) providing guidance for constructing the application ontology, and (ii) providing background knowledge for specifying the global control regime when the design model is constructed. For the first of these purposes, QUITE allows users to specify ontology mappings between STModel components and components of the application ontology. The ontology mapping editors are described in Section 5.2.2. For the second purpose, QUITE has extensive documentation facilities: every STModel, knowledge role, inference and mapping in the task model can be documented individually. Figure 19 summarizes the functionality provided by QUITE.

5.2.2. Quote

A second tool of the CUE environment is QUOTE. This tool is intended to support the development of application ontologies, either from scratch or by fine-tuning ontological theories selected from the ontology library described in Section 3. Further, the tool can be used for the development of ontological theories for the library.

Levels of support in QUOTE. QUOTE supports the definition of ontologies at three levels. The first of these, the ontology level, has to do with the selection of theories from a library to build an application ontology. The graphical interface enables users to get a quick overview of the contents of an ontological theory, without an analysis of the internal details of the definitions.

The second level of support is the theory level. QUOTE graphically shows the type constraints that are specified in the definitions of a theory. Whenever a parameter of a relation or a function is not typed, a warning is given.† QUOTE also warns the user

† In principle, nothing is wrong with untyped parameters. Neither Ontolingua nor QUOTE enforce such typing. However, the ontologies defined in QUOTE are intended for driving the knowledge-elicitation process, and type constraints on parameters are important for the validation of elicited knowledge.

Summary of QUITE's functionality

- **Supported activities from Section 5.1**
  - —2. Build an initial task model.
  - —3. Specify which parts of the task must be automated.
  - —5. Specify the role mappings.
  - —6. Map task model onto ontology.
- **Intended users**
  - —Knowledge engineers, in cooperation with domain experts.
- **Support types**
  - —Editing facilities.
  - —Library support.
- **Input from other tools**
  - —For the construction of the task model, QUITE requires no input from other tools.
  - —For the mapping between the task model and the application ontology, QUITE requires an ontology defined in Ontolingua.
- **Output**
  - —A task model.
  - —A series of ontology mappings.
- **Theoretical background**
  - —Task models can be constructed by configuring STModels for the three generic tasks in medicine: diagnosis, therapy planning and patient monitoring.

FIGURE 19. A synopsis of QUITE's functionality.

when definitions refer to classes, relations or functions that are not defined in the theory or its included theories; or when concepts are defined more than once.

The third level of support has QUOTE provides is at the level of Ontolingua definitions. The definition editors facilitate the definition of classes, relations and functions by syntax checking, automatic indentation, and by providing direct access to relevant parts of the on-line documentation distributed with Ontolingua. Therefore, every definition that is created or modified in QUOTE is guaranteed to be consistent with the Ontolingua language definition. QUOTE works directly on Ontolingua theories. Theories that are created using QUOTE are saved as Ontolingua files, and thus can be used directly as input for the Ontolingua translators described in (Gruber, 1993). Furthermore, QUOTE can also be used to edit or visualize Ontolingua files that were not created with the tool. For these reasons the tool can also be used outside the CUE framework.

QUOTE's functionality. The functionality of QUOTE will be demonstrated by showing how the tool supports the development of a simple application ontology in the domain of managing graft-versus-host disease (GVHD).

When QUOTE is started, the first window that appears is the *theory-inclusion-graph viewer* shown in Figure 20. This facility can be used to select and load theories from the library. The loaded theories are automatically added to the application ontology. The theory-inclusion-graph viewer visualizes the theories that are part of the application ontology and their inclusion relations. As explained in Section 3, a theory includes another theory when the definitions in the former depend on definitions in the latter. For example, the theory finding, which contains the definition of the concept `finding`, includes the theory observable, which defines the concept `observable`, because findings are defined as expressions about

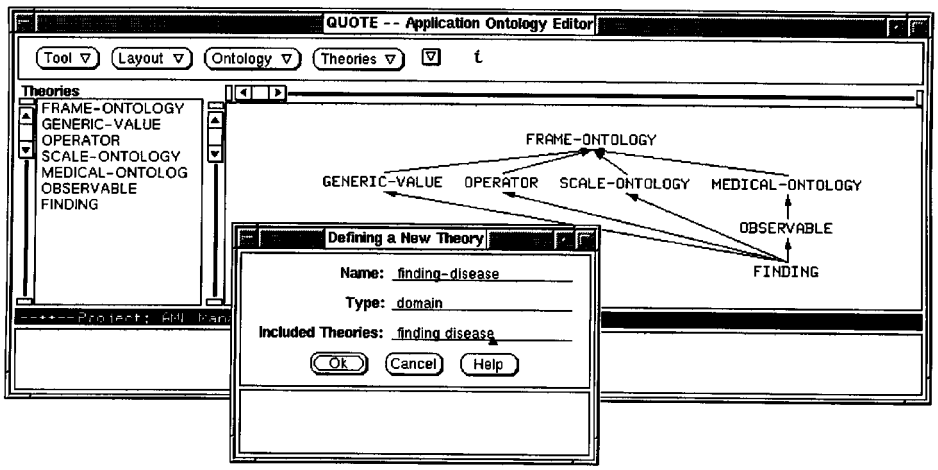

FIGURE 20. A visualization of the theory structure of an application ontology in QUOTE. The arrows indicate direct inclusion relations. When the user presses the OK button, the theory finding-disease is added to the graph.

observables. All theories that are part of the inclusion graph in Figure 20 are loaded from the library of ontological theories.

In Figure 20, the user is defining a new theory, finding-disease, in which the concepts will be defined that specify how diseases are related to findings in the GVHD domain.† Since the definitions of these concepts depend on the definitions of findings and of diseases, the theories that contain these definitions are included in the new theory.

When the new theory is created, it is automatically added to the theory inclusion graph. The contents of theories can be specified or altered by means of *theory editors.* The user interface of a theory editor consists of two areas (see Figure 21).‡ The upper area contains a number of browsers which show the classes, relations and functions that are defined in the theory, and one which shows the theories that are imported by the theory. The lower area shows a graphical representation of the structure of the theory. The rounded boxes in the lower area of the tool represent already defined classes, and the rectangular boxes represent defined relations. The texts `finding-importance`, `evoking-strength` and `frequency` represent functions.

QUOTE distinguishes between classes that are only intensionally defined and classes for which the instances are enumerated in the definition. We call the latter enumerated classes. The difference between these two types of classes is important because it affects the knowledge acquisition process: the definition of instances of enumerated classes is part of application ontology construction, whereas the definition of instances of intentionally defined classes is part of the model instantiation activity. Defining an enumerated class is a way to prevent CUE from

† In reality, finding-disease is also part of the ontology library. We assume here that it must be defined by the knowledge engineer to illustrate the functionality of QUOTE.

‡ Actually, there are three areas since every CUE tool has a feedback area at the bottom of the tool. This area is used to provide the users with feedback about the actions that they initiate.

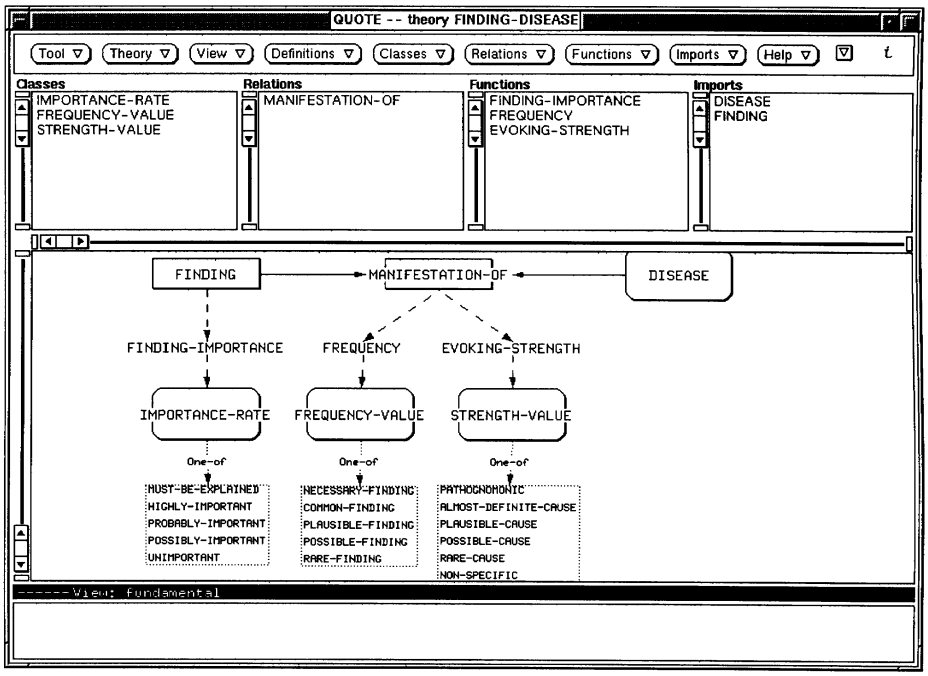

FIGURE 21. QUOTE's theory editor visualizing the theory finding-disease. Note that the terms "disease" and "finding" in the imports browser refer to the theories included by finding-disease. In these theories the concepts `finding` and `disease` are defined.

attempting to elicit other instances of that class during the model instantiation activity. A typical use of enumerated classes is for defining value sets for attributes. For enumerated classes, QUOTE also shows the instances. For example, in Figure 21 the instances of the enumerated classes `importance-rate`, `strength-value` and `frequency-value` are displayed. The arrows in the graph indicate type constraints. For instance, the relation `manifestation-of` shown in the figure is defined to have a disease and a finding as its arguments.

The user can edit the definitions by opening a *definition editor* on a relation, class or function. Definition editors allow modification of the definitions at the Ontolingua level. For instance, in Figure 22 the user has opened a definition editor for the function `frequency`. Definition editors are text buffers which provide emacs-like editing facilities. The CUE architecture ensures that the graphical representation and the underlying Ontolingua definitions are always consistent. This architecture is based on a user-interface management system which detects user actions and propagates these to the data structures, and which automatically detects changes in the data structures and propagates these to the visualizations. A detailed description of this architecture can be found in Wielemaker and Anjewierden (1989) and Anjewierden *et al.* (1992*a*).

In Section 4 three types of support were identified for ontology construction; specialized editing facilities, library support and process support. Only the first two of these support types are provided by QUOTE. The specialized editing support is based on three types of functionality: syntax checking, type checking and graphical

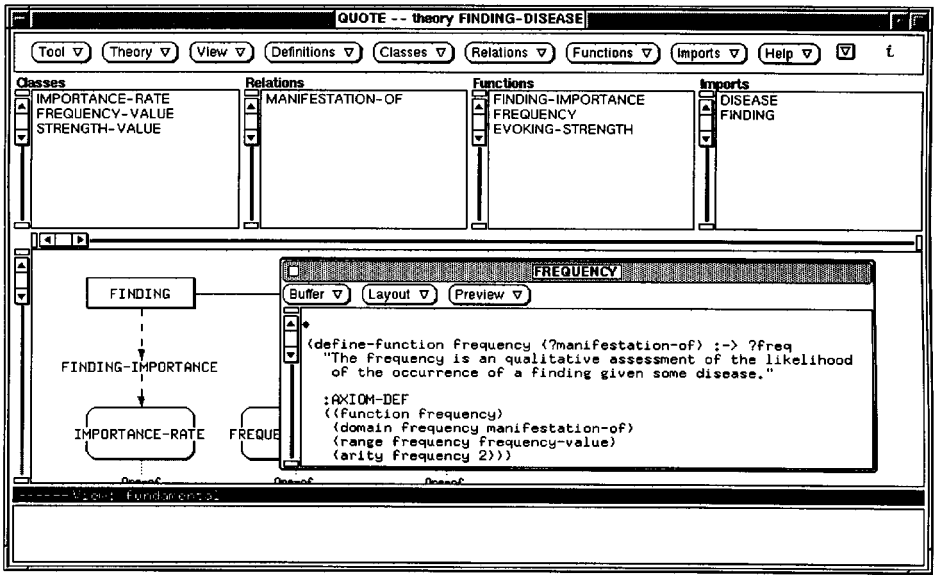

FIGURE 22. QUOTE's theory editor when the user is editing the function `frequency`. The window labelled "frequency" in the lower area of the theory editor is an example of a definition editor.

visualization. Although these functionalities facilitate the definition of ontologies significantly, the support remains passive: the user defines the concept, and the tool warns that something might be wrong or missing. The creative aspect of ontology construction remains a task for the user. However, the ontology library ensures that in many cases application ontology construction is reduced to library selection. Figure 23 summarizes the functionality provided by QUOTE.

Summary of QUOTE's functionality

- **Supported activities from Section 5.1.**
  —4. Construct the application ontology by selection, editing and configuration.
- **Intended users**
  —Knowledge engineers, in cooperation with domain experts.
- **Support types**
  —Editing facilities.
  —Library support.
- **Input from other tools**
  —QUOTE does not require input from other tools, but it can be used to visualize or edit Ontolingua theories developed outside the CUE environment. QUOTE can best be used together with a library of ontological theories, but this is not a requirement.
- **Output**
  —An application ontology, consisting of a number of Ontolingua theories.
- **Theoretical background**
  —The theory that underlies QUOTE is the Ontolingua theory: ontologies consist of definitions of classes, relations and functions that are organized in theories. A definition consists of a set of labelled sentences.

FIGURE 23. A synopsis of QUOTE's functionality.

5.2.3. Connecting task model and application ontology

Once the task model and the application ontology have been developed, they must be connected. This can be done using QUITE which has specialized mapping editors for supporting this activity. Every role and every inference in the task model can be associated with ontology mappings. For knowledge roles, these mappings specify which domain concepts may play these roles. For example, it may be specified that instances of the ontological class disease may play the role of diagnostic hypotheses. For, inferences, the mapping specify the types of domain knowledge that are used to perform the inference step. Thus, in the case of inferences, the mappings specify how the ontological requirements of the problem-solving methods associated with the inferences are satisfied. The screen dump of QUITE in Figure 46 shows two mapping editors.

5.3. MODEL INSTANTIATION IN CUE

Skeletal models specify which kinds of knowledge are needed for applications, and how the knowledge will be used during reasoning. The purpose of QUAKE, CUE's model instantiation tool, is to interact with the domain expert to collect the domain knowledge and store it in a knowledge repository. In Section 4, five types of support were identified for model instantiation: (i) consistency checking, (ii) completeness checking, (iii) use of domain specific terminology, (iv) use of intuitive visualization and (v) dialogue structuring. This section describes how each of these forms of support are provided by CUE.

QUAKE can be used in two modes of interaction: passive, where the user determines the structure of the knowledge elicitation dialogue, and active, where the tool acts as an interviewer. Section 5.3.1 describes how QUAKE can be used in passive mode, thereby illustrating how the tool performs consistency checking and how it uses domain specific terminology. Section 5.3.2 describes the active mode, in which the tool also checks for completeness and structures the knowledge elicitation dialogue. Section 5.3.3 describes the use of specialized visualization techniques in QUAKE. Finally, Section 5.3.4 explains how QUAKE exploits the application ontology to support model instantiation.

5.3.1. QUAKE as a passive consistency checker

QUAKE provides a narrow view on the underlying knowledge base. Only parts directly relevant to the current elicitation activity are shown. QUAKE's basic user interface, shown in Figure 24, consists of three areas. The upper left area is the object window. An object is either an instance of a class or a tuple of a relation. The object window is used for displaying information about the object that is the focus of the current elicitation activity. The right upper area contains a multi-functional browser. Depending on the nature of the elicitation activity, this browser can show different types of objects. The lower area of the tool is the interaction window. In this window the user is prompted to assert new knowledge in the knowledge repository.

The use of QUAKE in passive mode will be illustrated with a fragment of a knowledge-elicitation scenario for a system that diagnoses GVHD. For this scenario, we use the application ontology described in Section 5.2.2. The scenario will also be used in Section 5.3.2 to describe some more advanced features of the tool.

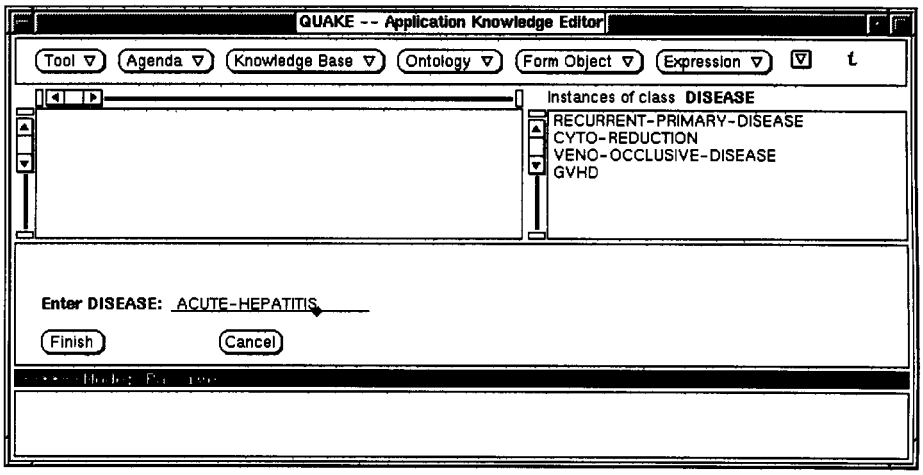

FIGURE 24. QUAKE after the domain expert has entered some diseases. The entered diseases are visualized in the browser on the right side of the tool. The user is just entering a fifth disease: ACUTE-HEPATITIS.

A knowledge-elicitation scenario. In the example scenario, the domain expert starts the knowledge elicitation session by entering diseases. Therefore, the user focuses the tool on the class `disease`, which turns the multi-functional browser into a browser for disease instances. The expert enters the names of some diseases that are relevant in the application area. The result is shown in Figure 24.

Once five diseases have been entered, the user decides to concentrate on one of them: GVHD. The disease is selected and visualized in QUAKE's object window. In the application ontology, no attributes are defined on instances of class disease. Therefore, the user decides to ask the tool for the relations that are defined on diseases. According to the application ontology, there are three kinds of relations defined on instances of the class disease: `disease-subtype`,† `manifestation-of` and `has-treatment`. The relations are shown in the browser, which is now used as a relation browser. In Figure 25, the relation `disease-subtype` is mentioned twice, because GVHD can play two roles in this relation. In the first relation specifier GVHD plays the role of the supertype, whereas in the second specifier GVHD would be the sub-type.

The domain expert decides to work first on the manifestations of GVHD, so (s)he selects that relation in the browser. The tool responds by showing all the findings that are defined as manifestiations of GVHD. However, in this case there are as yet no findings associated with GVHD. The domain expert decides to enter the presence of a rash as a manifestation of GVHD and selects the corresponding pulldown option, which results in the tool showing a template for the `manifestation-of` relation in the interaction window. Because the domain expert is working on GVHD, the disease parameter is already instantiated. As illustrated in Figure 26, the user enters the finding "rash = present" in the text field. When the

† Note that the disease-sub-type relation is an object level relation that can hold between *instances* of the ontological class disease. This relation has nothing to do with the sub-class relation which is used in Ontolingua.

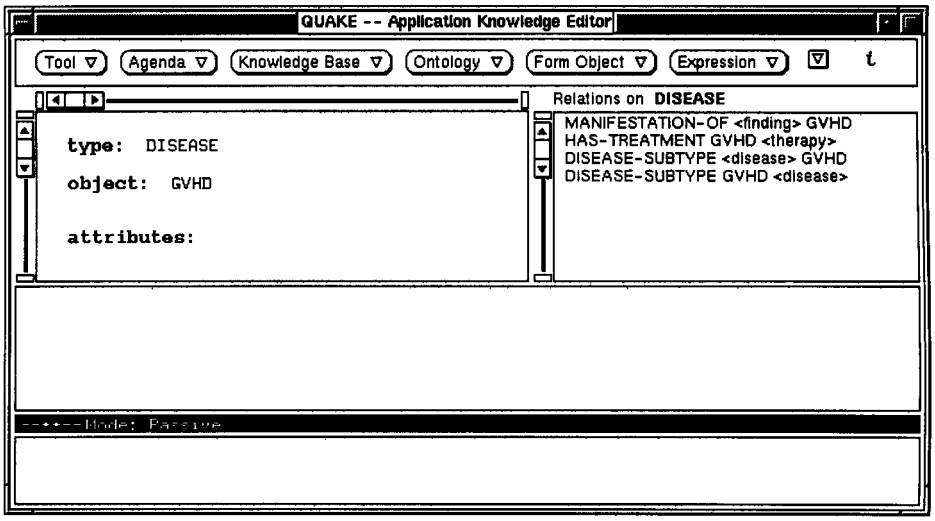

FIGURE 25. QUAKE showing the relations defined on the class of GVHD.

domain expert is finished QUAKE checks whether the entered expression is syntactically correct and consistent with the application ontology. If the new expression is correct and does not conflict with previously entered information, it is asserted in the QUAKE knowledge base. An example of a possible conflict would be that the domain expert had already asserted that `rash` is an instance of disease. Since it is defined in the application ontology that a `finding` has an `observable` as its first parameter and `disease` is not specified as a sub-class of `observable` or vice versa, QUAKE would in that case refuse to accept the entered finding.

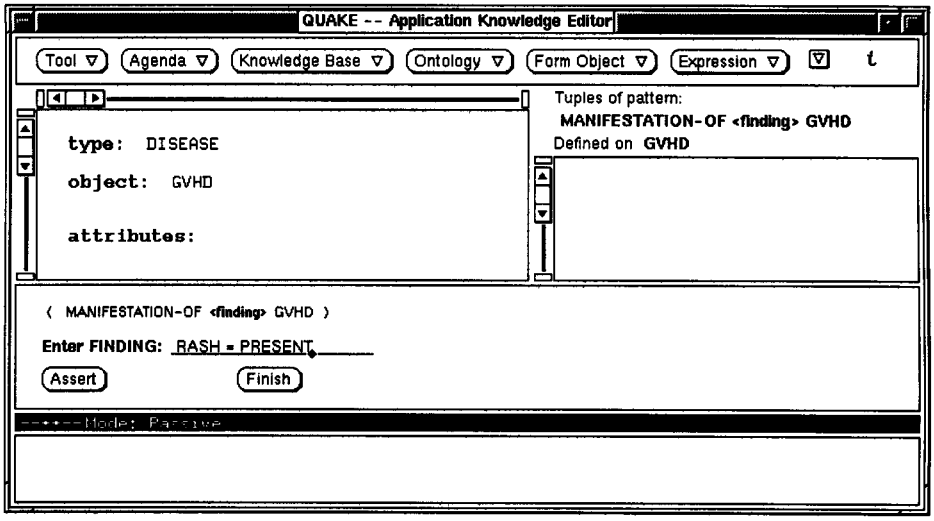

FIGURE 26. QUAKE when the domain expert enters that the presence of rash is a manifestation of GVHD. When the user does not know in which form the finding should be entered the tool can be asked to show a template of the relation.

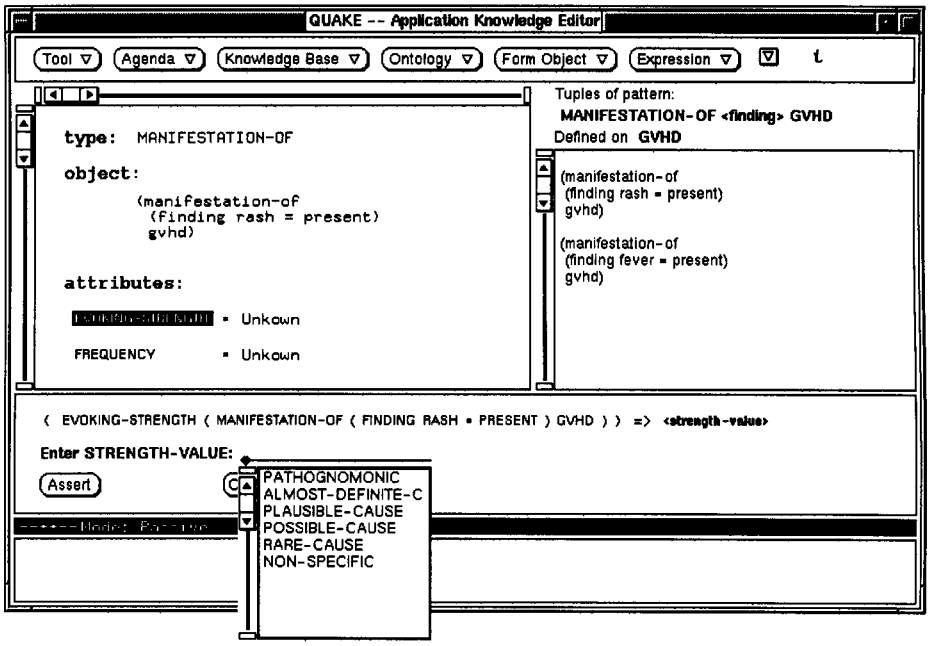

FIGURE 27. QUAKE when the domain expert enters the evoking strength of the presence of rash for GVHD. Because strength-value is an enumerated class, the tool is able to show the allowed values.

In the scenario, the domain expert continues by asserting that another manifestation of GVHD is the presence of fever. After that (s)he decides to specify some further qualifications of the first mentioned manifestation. The corresponding tuple is selected in the browser and displayed in the upper left window. According to the application ontology, `manifestation-of` relations have two attributes: `evoking-strength` and `frequency`.† The user first selects the `evoking-strength` attribute and as a result a template for the function appears in the interaction window (Figure 27). Because in the application ontology `evoking-strength` is defined to have a `strength-value` as its value, which is an enumerated class, QUAKE is able to generate the list of possible values for the attribute. This list is used to support an auto-completion facility (which is also displayed in Figure 27). This example clearly illustrates the importance of the distinction between intensionally defined and enumerated classes mentioned in Section 5.2.2: in the model instantiation phase it is not possible to define instances of enumerated classes.

The scenario shows that QUAKE uses the application ontology to provide strong guidance for the model instantiation process. The tool prevents the user from entering expressions that conflict with the definitions in the ontology, and the tool interacts with the user in domain oriented terminology: it prompts for diseases and findings, and not for method-specific knowledge types, such as hypotheses and data, or for symbol-level constructs such as rules or constraints. Of course, the ability of the tool to communicate in domain-specific terminology depends on the domain

† QUAKE interprets unary functions as attributes.

specificity of the application ontology. If a knowledge engineer uses only generic vocabulary in the definitions, QUAKE is only able to communicate in generic terms. The knowledge engineer should take this into account when constructing the application ontology.

QUAKE confronts the user with a limited amount of information at a time. For instance, Figure 27 shows only the manifestations of GVHD, and the evoking-strength and the frequency for one of these manifestations. The rationale behind this approach is that this narrow, focused view on the underlying knowledge base guards the domain expert from not seeing the wood for the trees. Experiences with earlier KA workbenches showed that domain experts often get confused when large amounts of heterogeneous domain facts are displayed at one time.

5.3.2. QUAKE as an active knowledge collector

Experience with QUAKE as a passive application knowledge editor has revealed some shortcomings. Because of QUAKE's narrow view on the knowledge base, users quickly forget which knowledge has already been asserted and which knowledge still must be entered. For example, in the scenario described in the previous section, the user first entered five diseases, then (s)he concentrated on one of these, GVHD, and asked the tool which relations were defined on this disease. Of the four relations, `manifestation-of` was selected, and two tuples of this relation were entered. In the course of this scenario many tasks were left unfinished. For instance, besides the five diseases shown in Figure 24 other diseases need to be entered which also have findings as manifestations. Further, the disease hierarchies (the `disease-subtype` tuples) must be specified, etc. In passive mode, QUAKE leaves the navigation in the knowledge space defined by the skeletal model completely to the user.

To overcome this difficulty, QUAKE is also equipped with a more active interaction style. In active mode, the tool not only waits for the user to take action, but it can also take the initiative. The active component of the tool consists of two parts: (i) an *agenda mechanism,* responsible for completeness checking and (ii) an interpreter for *knowledge-elicitation strategies,* responsible for dialogue structuring.

Agenda mechanism. The purpose of the agenda mechanism is to keep a record of which parts of the skeletal model are fully instantiated, partially instantiated, or empty. In some cases, QUAKE can decide whether a part of the skeletal model has been fully instantiated. For example, one of the assumptions made by QUAKE is that attributes must always have values. Therefore, the tool can decide that a particular attribute still needs to be specified without intruding upon the user. Furthermore, sometimes the application ontology explicitly defines that a specific number of relation tuples or class instances must exist. For instance, it could be defined that instances of some type of disease may have at most one therapy. QUAKE's agenda manager can use such information to decide whether parts of the skeletal model are fully instantiated or not.

Most of the time, however, the decision as to whether a part of the skeletal model is fully instantiated must be taken by the domain expert. In the scenario, it was the domain expert who had to decide that all the relevant diseases were entered, that all the manifestations of all the diseases were specified, etc. In contrast, in active mode it is up to QUAKE to keep the agenda up to date. Whenever a user decides to start working on another part of the knowledge base and the tool cannot determine by

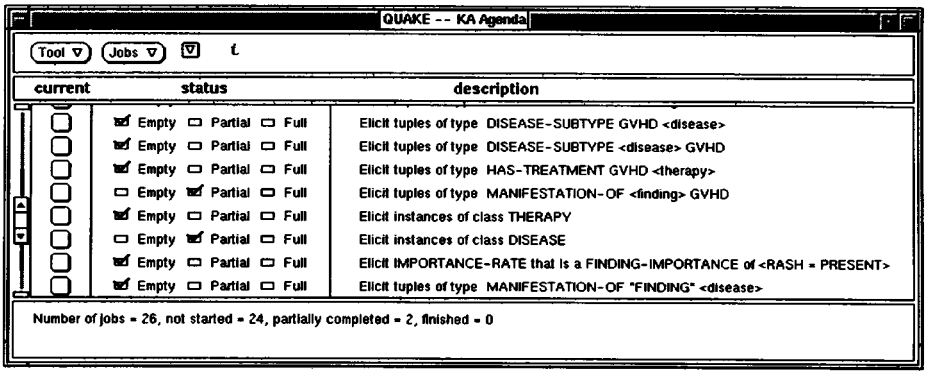

FIGURE 28. QUAKE's agenda mechanism.

itself that the job that was worked on has been completed, QUOTE asks the user. For example, when the user in the scenario in Section 5.3.1 decided to start working on the manifestations of GVHD, the tool in active mode would first have asked whether the entered diseases are all the relevant diseases in the application domain. The response of the user would then be used to update the agenda. Figure 28 shows QUAKE's agenda after the five diseases and the two manifestations were entered.

Knowledge elicitation strategies. The agenda mechanism maintains a list of knowledge-elicitation activities that are completed, partially completed, or not yet initiated. However, the decision as to the order in which the different elicitation activities are performed is still left to the user. For example, in the scenario in Section 5.3.1 it was the user who decided to start working on the diseases, and it was the user who decided to select the `manifestation-of` relation from the relations defined on GVHD. After a while, the decision as to which knowledge-elicitation activity should be performed next becomes a complicated task in itself, because the number of activities rapidly increases as new knowledge is entered. For example, in the above scenario, four activities are added to the agenda for every disease which is entered in the knowledge base (elicitation of the manifestations, treatments, sub-types and supertypes of the entered disease).

Many knowledge acquisition tools that are specialized in the model instantiation activity of the knowledge acquisition process, take a more active role. For example, MOLE (Eshelman, 1988) and SALT (Marcus & McDermott, 1989) instantiate their skeletal models using a dialogue where the system takes the initiative. In these systems, the tool decides which knowledge should be elicited when. We call the structuring principles for such a dialogue a knowledge-elicitation strategy. In other words, a knowledge-elicitation strategy is a specification of the order in which the domain instances and expressions are to be elicited.

In the above-mentioned second-generation KA tools, it was possible to hardwire the knowledge-elicitation strategies in the program because these tools were based on a fixed skeletal model. MOLE for example, begins a knowledge-elicitation session by asking the user to list some of the complaints that would indicate that there is a problem to be diagnosed. After these are entered, the tool asks for states or events that explain these complaints. In turn these states may also need to be explained. In

this way MOLE builds, in a breadth-first manner, a network of causally related states and events. MOLE derives its power from the strong assumptions that it makes about the structure of the causal network that is required for the Cover-and-Differentiate problem-solving method.

It is not possible to use built-in knowledge-elicitation strategies in systems like CUE, where the construction of the skeletal model is also considered part of the knowledge acquisition process. Because the suitability of a strategy depends heavily on the nature of the skeletal model, the strategy can be determined only after the skeletal model has been constructed. In the current version of CUE this problem is addressed by making the specification of the knowledge-elicitation strategy part of the construction of the skeletal model, as is also the case in DIDS (Runkel & Birmingham, 1994) and KREST (Steels, 1993).

To formulate knowledge-elicitation strategies as part of the skeletal model, a simple Lisp-based language has been defined which can be interpreted by QUAKE. This language allows the knowledge engineer to express ordering constraints in terms of the application ontology. A very simple example of a knowledge-elicitation strategy defined in this language is the following.

```
(define-ka-strategy main ()
  (elicit-all ?d (disease ?d))                                    (1)
  (for-each ?d (disease ?d)                                       (2)
    (elicit-all $f (manifestation-of (finding $f)
                                     (disease ?d)))))
```

The first expression tells QUAKE to start eliciting all the diseases in the domain. The second specifies that, once the diseases have been elicited, all the manifestations for each disease must be elicited. The constructs `elicit-all` and `for-each` are the main primitives of the language. `elicit-all` takes a specification and tells QUAKE to elicit expressions that are in accordance with that specification. In the body of the construct, operations can be specified that must be performed on each of the elicited expressions. `for-each` works similar, but instead of eliciting expressions according to the specification, it retrieves expressions that are already stored in QUAKE's knowledge repository. The language is extended with constructs for sequencing, iteration and simple conditionals and allows recursion. The suitability of the language has been tested by using it for specifying the knowledge elicitation strategies of MOLE and SALT.

The use of a language for this purpose allows specification of any knowledge-elicitation strategy which can be defined in ontological terms. It is left to the knowledge engineer to decide which of these strategies are sensible. This is an undesirable situation because it makes the job of the knowledge engineer more difficult. What is really needed is a KA tool that is able to determine an appropriate knowledge-elicitation strategy by itself. Such a tool would need to have knowledge of general guidelines for the formulation of knowledge-elicitation strategies. At present, such general principles are not available. In the remainder of this section, some candidate principles on which knowledge elicitation strategies can be based are discussed.

Principles for structuring the KA dialogue. To investigate the question as to whether which dialogue structuring principles are sensible, we have compared the elicitation

strategies employed by a number of existing tools. For one of these tools, MOLE, we have already described its elicitation strategy. SALT constructs its knowledge base in a similar way. The skeletal model of this tool requires three types of knowledge: procedures, constraints and fixes. The knowledge pieces are organized in a dependency network with three types of relations: contributes-to, constrains, and suggests-revision-of. To instantiate this skeletal model, the tool allows the user to start elicitation at any point in the network. SALT then cues the user for appropriate links and keeps track of how the elicited knowledge pieces are fitting together and it also warns for inconsistencies.

ALTO (Major & Reichgelt, 1990) is a tool for the elicitation of concept hierarchies, based on the laddering technique. The underlying skeletal model distinguishes two types of knowledge: concepts, which are organized in `is-a` hierarchies, and attributes of those concepts. ALTO starts the elicitation process by asking for a seed item. From this seed item, the user may move up or down the hierarchy, or to the siblings of the seed item. After that, the attributes of the new concept are elicited and the process continues with the elicited concept as the new seed item.

Analysis of the three knowledge-elicitation strategies described above reveals some striking similarities. All these tools seem to do some kind of ''graph traversal''. This is one example of a potentially general principle for formulating knowledge-elicitation strategies: use elicited pieces of knowledge to prompt for related pieces of knowledge. This principle may be applied in a depth-first manner, a breadth-first manner, or a combination of both and it can make use of multiple relations (e.g. ''contributes-to'' and ''constraints'' in SALT).

A second general principle is based on the observation that in many application domains, there are ''basic'' objects. The nature of these objects depends on the application task. For example, in diagnostic applications, the basic objects are the diagnoses, whereas in design applications, the basic objects are components. This observation is for example used in KEW's advice and guidance module to organize the KA process. When the task of the application is of a diagnostic nature, KEW's task scheduler suggests starting by eliciting the potential *solutions.* In cases when the number of solutions is infinite, or very large, KEW instead suggests starting by eliciting the *solution components.*

Contrary to the first principle, the second principle is dependent on the task. The ability of QUAKE to implement such strategies depends on the mapping between the task model and the application ontology. For example, in the GVHD domain, QUAKE should know that the potential solutions for the diagnostic problem are diseases. In general, this heuristic could be formulated as follows. In every application, there are basic objects. What these basic objects are depends on the nature of the task and on the mapping between the task model and the application ontology. Knowledge acquisition should start by eliciting these basic objects.

In summary, based on an analysis of the strategies used in existing knowledge acquisition tools, the following dialogue-structuring principles can be formulated.

- Use already elicited knowledge to prompt for related pieces of knowledge (graph traversal).
- Center elicitation around ''basic objects''. Which objects are basic depends on the task type and the mapping between the task model and the application ontology.

Principles such as these can be used as global constraints that elicitation strategies should satisfy. However, they are not sufficiently restrictive to derive a single best elicitation strategy from a task model and an application ontology. Further, the optimal strategy could also depend on other, not ontology related considerations such as the level of expertise of experts and the level of experience with tools (Burton, Shadbolt, Rugg & Hedgecock, 1990).

5.3.3. Specialized visualization in QUAKE

In the sections above it was illustrated how QUAKE checks for consistency and how it uses domain specific terminology (Section 5.3.1) and how it checks for completeness and structures the knowledge elicitation dialogue (Section 5.3.2). The final way in which tools can support model instantiation is by the use of specialized visualizations. ALTO, for instance, visualizes the elicited concept hierarchies in the form of directed graphs. The tool has this ability because it makes ontological assumptions about the structure of the knowledge that needs to be elicited. In ALTO these assumptions are hard-wired in the tool. In contrast, QUAKE cannot make such ontological assumptions because it must be able to instantiate arbitrary ontologies defined with QUOTE.

In order to use specialized visualizations for particular parts of the knowledge base, QUAKE must be able to determine on the fly which visualizations are appropriate for which parts of the knowledge base. One way to realize this is to explicate the ontological assumptions upon which specialized visualizations are based. This makes it possible to decide whether a particular class or relation can be displayed using a specialized visualization, based on the definition of that class or relation in the application ontology. Currently, the only specialized visualization facility provided by QUAKE is a directed graph viewer. This viewer can be used for relations that are binary, transitive and anti-symmetric. Whenever QUOTE is able to determine that these properties hold for a certain relation, the tuples of this relation may be visualized using the directed graph viewer. Figure 29 shows this facility when visualizing tuples of the `disease-subtype` relation.

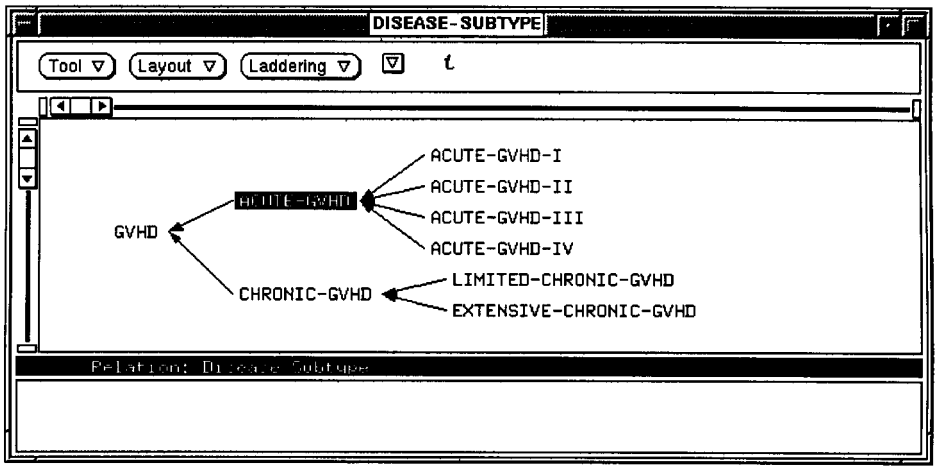

FIGURE 29. QUAKE's directed graph viewer, visualizing tuples of the `disease-subtype` relation.

Summary of QUAKE's functionality

- **Supported activities from Section 5.1**
  - —7. Create elicitation agenda.
  - —8. Specify knowledge-elicitation strategy.
  - —9. Elicit domain knowledge.
- **Intended users**
  - —Domain experts.
- **Support types**
  - —Consistency checking.
  - —Completeness checking.
  - —Use of domain specific terminology.
  - —Intuitive visualization.
  - —Dialogue structuring defined by knowledge engineer.
- **Input from other tools**
  - —QUAKE Requires an application ontology defined with QUOTE.
  - —When there is also a mapping between the task model and the application ontology QUAKE can use this to generate an initial agenda. Otherwise, the user will have to aid the tool.
- **Output**
  - —A complicated knowledge model, which can be handed over to a programmer to construct the design model.
- **Theoretical background**
  - —The theoretical background of QUAKE is basically the theory of model-based knowledge acquisition that was set out in Section 4: focused knowledge elicitation requires a restrictive skeletal model. In QUAKE, the skeletal model is made restrictive by incorporating the application ontology.

FIGURE 30. A synopsis of QUAKE's functionality.

The directed graph viewer can also be used as a simple laddering tool. To do this, the user must select an object in the hierarchy and select the ''Ladder Up'' or ''Ladder Down'' option (from the Laddering pulldown menu). The tool then searches for the corresponding job in the agenda and starts it up. Figure 30 summarizes the functionality provided by QUAKE.

In summary, the CUE tools support the steps distinguished in the generic scenario of Section 5.1 in the following ways.

(1) **Informally describe domain and task of the application.** CIE does not support this step.
(2) **Identify generic tasks.** In CUE, this step is supported by QUITE. This tool allows the user to select generic-task instances and to configure these into a task model (by mouse clicking and dragging).
(3) **Specify which parts of the task must be automated.** QUITE allows the user to indicate which parts of the task model are to be performed by the KBS.
(4) **Construct the application ontology.** QUOTE supports ontology construction by selecting ontological theories from a library, configuring the theories into an application ontology and refining the definitions in the theories for the particular application.
(5) **Specify the role-to-role mappings.** QUITE allows the user to specify the mappings between already defined roles (by dragging lines between the roles in the graphical representation of the task model).

(6) **Map task model onto ontology.** The task model can be mapped onto the ontology in QUITE via the use of ontology mapping editors.
(7) **Create elicitation agenda.** Before the elicitation activity starts, QUAKE generates an initial agenda which is automatically kept up to date during the elicitation session.
(8) **Specify knowledge-elicitation strategy.** This step is only partially supported. QUAKE provides a simple tailored language for specifying knowledge-elicitation strategies, but it does not have specialized editors to support the language.
(9) **Elicit domain knowledge.** This step is supported by QUAKE. The tool interprets the knowledge-elicitation strategy and prompts the user to enter new knowledge until the knowledge base is—according to the application ontology—complete. QUAKE checks for consistency and is able to select appropriate visualizations for parts of the knowledge base.

5.3.4. How it works

QUAKE's ability to support model instantiation is almost entirely based on its capacity to inspect Ontolingua definitions. Consistency checking in QUAKE means checking whether the entered piece of knowledge is consistent with the corresponding Ontolingua definition. Also, completeness checking in QUAKE (determining if a KA activity has been completed) is done by examining Ontolingua definitions to see how many instances or tuples of a particular class or relation are allowed. Further, to decide how knowledge pieces should be visualized, the corresponding definitions are inspected. This section describes the components of an Ontolingua definition and how QUAKE is able to inspect these.

Ontolingua definitions. An Ontolingua definition consists of a number of labelled sets of sentences that specify how the defined class, relation or function may be used. For our purposes, the most important distinction between these sets of sentences is that some sets consist of axioms in which the defined term is used, while other sets of sentences consist of meta descriptions of the defined terms. In the former, which are called first-order sentences in Ontolingua, the truth functional properties of the logical connectives are used to constrain under which circumstances the defined term can be used for formulating valid expressions. The other sets of sentences consist of meta descriptions of the defined terms. These expressions are called second-order sentences in Ontolingua. A simple example of a sentence of the first type is:

```
(⇒ (manifestation-of ?finding ?disease)                    (1)
   (disease ?disease))
```

An example of a sentence of the second type is;

```
(nth-domain manifestation-of 2 disease)                    (2)
```

The two sentences both express the fact that the second argument of `manifestation-of` must be a disease. However, while the first sentence expresses this fact only implicitly, using material implication, the second states the fact explicitly.†

† The terms implicit and explicit are used in the same way as in (Kirsh, 1990): something is explicitly represented when it can be derived from a knowledge base without the application of inference steps.

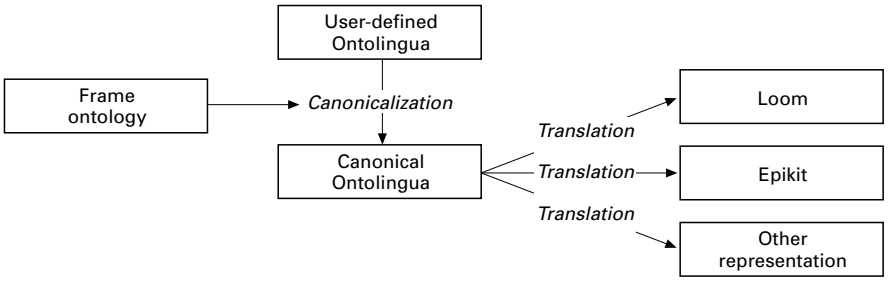

FIGURE 31. The Ontolingua translation architecture.

The vocabulary for writing these explicit meta axioms about ontological terms is defined in the Frame ontology, a special representational ontology that comes with Ontolingua. The Frame ontology is provided to facilitate the translation of Ontolingua ontologies into a number of different representation formalisms, which was the main purpose for developing Ontolingua. The idea is that the terms defined in the Frame ontology capture cliches for which many (frame-based) problem solvers provide specialized inference procedures. The translators are able to inspect the meta axioms for deciding how a particular definition should be translated.

In order to facilitate the job of the Ontolingua translators, the Ontolingua system first performs a canonicalization step. In this step, the system attempts to recognize first-order cliches and reformulate them in terms of the Frame-ontology axioms. The canonicalization pass guarantees that the translation output profits as much as possible from the special inference procedures of the target systems. Figure 31 shows the Ontolingua translation architecture. A detailed description of this architecture can be found in Gruber (1993).

QUOTE and Ontolingua. The definition editors that are provided by QUOTE for specifying ontology definitions provide emacs-like editing facilities such as parenthesis checking, automatic indentation, etc., but they are not syntax-driven. Users can use both types of sentences mentioned above in their definitions. Once they are satisfied with a definition, it is checked for syntax errors, and then handed over to the Ontolingua system. Ontolingua canonicalizes the definition entered by the user, and then passes it to a QUOTE-specific translator, which translates the definition into QUOTE's internal data structures. Then, QUOTE invokes its pretty-printer to produce a nicely formatted textual representation of the canonicalized definition in the definition editor. Figure 32 summarizes this process.

A result of this design is that the internal representation—and the visualized representation—of QUOTE are always equivalent to canonical Ontolingua, making maximal use of the axioms in the Frame-ontology. This is important because the other tools in CUE can inspect only the meta-axioms. For example, when QUAKE needs to know the type of the second argument to `manifestation-of` it would understand sentence 2 above, but it would not understand sentence 1. However, the architecture ensures that sentence 1 is automatically reformulated as sentence 2.

QUAKE and Ontolingua. Just as the Ontolingua translators are able to inspect the meta axioms in Ontolingua definitions to decide how a particular term should be translated, QUAKE is able to inspect the meta axioms to retrieve information needed

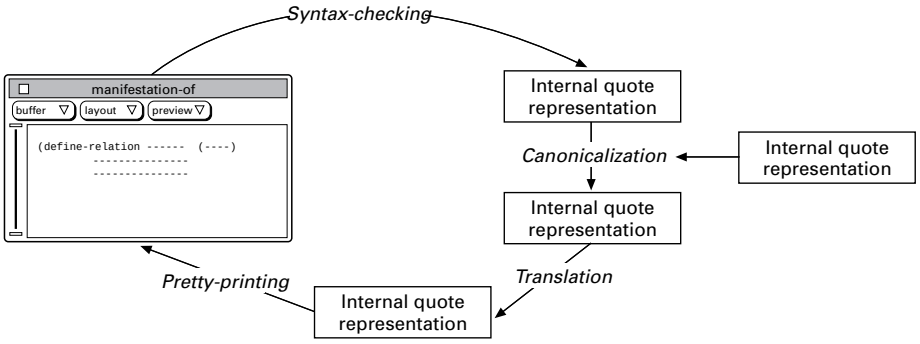

FIGURE 32. The interaction between QUOTE and the Ontolingua translation architecture.

to support knowledge elicitation. For example, in Section 5.3.3 it was mentioned that for visualizing the tuples of a relation as a directed graph, QUAKE requires that the relation is binary, transitive and anti-symmetric. Since each of these terms is defined in the Frame ontology, QUAKE only needs to inspect the meta axioms of the ontology definition of the relation to decide whether these properties hold. In the same way, meta axioms about the cardinality of relations can be used to decide whether a particular knowledge acquisition job has been completed. For example, when it is defined that the maximum-slot-cardinality of a particular class and a particular binary relation is three, QUAKE can decide that the corresponding KA job is finished after three tuples of that relation have been elicited.

QUAKE makes a KA-oriented interpretation of the concepts defined in the Frame ontology, in the same way that the target representations that Ontolingua translates to make a reasoning-oriented interpretation of these concepts. Of course, not every concept in the Frame ontology can be used for all types of model-instantiation support provided by QUAKE. Figure 33, which is based on the description of the Frame ontology by Gruber (1993), summarizes the kinds of information that can be derived from the terms defined in the Frame ontology. The information can be used for three purposes: consistency checking, completeness checking and visualization. The other types of support for model instantiation, domain-specific terminology and dialogue structuring, are not directly based on the inspection of the meta axioms in the definitions. The use of domain specific terminology comes for free as a result of the use of an explicitly defined application ontology, the dialogue structuring facilities are partially based on QUAKE's capacity for completeness checking.

5.4. CUE IN PERSPECTIVE

In this section we have presented three tools that are part of the CUE knowledge engineering workbench. CUE falls in the same category of knowledge acquisition environments as DIDS and PROTÉGÉ-II, which were described in Section 4. This section highlights some similarities and differences between CUE and these other systems.

The work must closely related to the work presented here is that on the PROTÉGÉ-II system. In particular, the work on DASH (Eriksson, Puerta & Musen, 1994) is in a similar spirit. DASH is a tool that can be used to build knowledge elicitation tools from application ontologies specified in MODEL, the PROTÉGÉ-II ontology language. The relation between QUAKE and DASH is similar to that between an interpreter and

Type	Relation	Arguments	Cons.	Comp.	Vis.
c	**relation**	?relation	−	−	+
c	**function**	?function	−	−	+
c	**class**	?class	−	−	+
r	**instance-of**	?individual ?class	−	−	−
f	**all-instances**	?class :→ ?set-of-instances	+	−	−
f	**one-of**	@instances :→ ?class	+	−	−
r	**subclass-of**	?class ?class	+	−	−
r	**superclass-of**	?class ?class	+	−	−
r	**subrelation-of**	?relation ?relation	+	−	−
r	**direct-instance-of**	?individual ?class	+	−	−
r	**direct-subclass-of**	?class ?class	+	−	−
f	**arity**	?relation :→ ?n	+	−	−
f	**exact-domain**	?relation :→ ?relation	+	o	−
f	**exact-range**	?relation :→ ?class	+	o	−
r	**total-on**	?relation ?relation	+	o	−
r	**onto**	?relation ?range-class	+	o	−
c	**n-ary-relation**	?relation	−	−	+
c	**unary-relation**	?relation	−	−	+
c	**binary-relation**	?relation	−	−	+
c	**single-valued**	?binary-relation	+	+	−
f	**inverse**	?binary-rel :→ ?binary-rel	o	−	−
f	**projection**	?relation ?column	−	−	−
f	**composition**	?rel1 ?rel2 :→ ?binary-rel	−	−	−
r	**composition-of**	?binary-rel ?list-of-rels	−	−	−
f	**compose**	@binary-rels ?binary-rel	−	−	−
r	**alias**	?rel1 ?rel2	−	−	−
r	**domain**	?relation ?class	+	−	−
r	**domain-of**	?class ?relation	+	−	−
r	**range**	?relation ?class	+	−	−
r	**range-of**	?class ?relation	+	−	−
r	**nth-domain**	?rel ?integer ?class	+	−	−
r	**has-value**	?inst ?binary-rel ?value	−	−	−
f	**all-values**	?inst ?binary-rel	−	−	−
r	**value-type**	?inst ?binary-rel ?class	−	−	−
f	**value-cardinality**	?inst ?binary-rel :→ ?n	−	−	−
r	**same-values**	?inst ?rel1 'rel2	−	−	−
r	**inherited-slot-value**	?class ?binary-rel ?value	−	−	−
f	**all-inherited-slot-values**	?class ?binary-rel :→ ?values	+	−	−
r	**slot-value-type**	?class ?binary-rel ?class	+	−	−
f	**slot-cardinality**	?class ?binary-rel :→ ?n	−	+	−
r	**minimum-slot-cardinality**	?class ?binary-rel ?n	−	+	−
r	**maximum-slot-cardinality**	?class ?binary-rel ?n	−	+	−
r	**single-valued-slot**	?class ?binary-rel	−	+	−
r	**same-slot-values**	?class ?rel1 ?rel2	+	−	−
c	**class-partition**	?set-of-classes	+	−	−
r	**subclass-partition**	?class ?class-partition	+	−	−
r	**exhaustive-subclass-partition**	?class ?class-partition	+	−	−
c	**asymmetric-relation**	?binary-relation	−	−	−
c	**antisymmetric-relation**	?binary-relation	+	−	+
c	**antireflexive-relation**	?binary-relation	+	−	+
c	**irreflexive-relation**	?binary-relation	−	−	−
c	**reflexive-relation**	?binary-relation	−	−	−
c	**symmetric-relation**	?binary-relation	−	−	−
c	**transitive-relation**	?binary-relation	−	−	+
c	**weak-transitive-relation**	?binary-relation	−	−	o
c	**one-to-one-relation**	?binary-relation	+	−	−
c	**many-to-one-relation**	?binary-relation	+	−	−
c	**one-to-many-relation**	?binary-relation	+	−	−
c	**many-to-many-relation**	?binary-relation	−	−	−
c	**equivalence-relation**	?binary-relation	o	−	−
c	**partial-order-relation**	?binary-relation	−	−	o
c	**total-order-relation**	?binary-relation	−	−	o
r	**documentation**	?object ?string	−	−	−

FIGURE 33. A summary of how QUAKE uses terms defined in the Frame ontology, to support consistency checking (cons.), completeness checking (comp.) and specialized visualization (vis.). +: QUAKE uses the relation for a type of support; o: the relation could be used but that the current implementation does not; c: class; r: relation; f: function; :→: separates domain parameters from range parameters for functions. Parameters starting with @ may be bound to multiple arguments.

a compiler. Whereas QUAKE interprets application ontologies to communicate in domain specific terminology, DASH uses these ontologies to generate other tools that communicate in domain specific terminology. DASH-generated tools act as user-friendly front-ends for the underlying knowledge base. They are similar to QUAKE in passive model in that they do not aim for completeness. That is, they do not actively search for missing information. Therefore it is to be expected that users of DASH generated tools will face the same problems as those encountered with QUAKE when used in passive mode.

The DIDS system (Runkel & Birmingham, 1994) has a facility for specifying knowledge-elicitation strategies. Runkel and Birmingham distinguish two elements that drive knowledge elicitation, namely (i) ''mechanisms for knowledge acquisition'' (MeKA), which define for each knowledge construct in the ontology an elicitation, a verification and a generalization procedure, and (ii) a ''knowledge acquisition method'' which defines the sequencing of MeKAs. The MeKA's are knowledge acquisition tools that are specialized for particular types of knowledge. This similar to the use of specialized visualizations in CUE. The knowledge-acquisition method is similar to knowledge-elicitation strategies in CUE, although they are typically more coarse grained.

The main contribution of the work on CUE is that it provides a theoretical foundation for how the different types of support for knowledge elicitation can be achieved by separating ontology and application knowledge. If the ontology is available, it can be inspected to check the elicited knowledge for consistency and completeness, to communicate with the expert in domain specific terminology and to choose suitable visualizations. It was hypothesized that also a fifth type of support could be derived from the ontology: dialogue structuring. To experiment with different dialogue structuring principles CUE is equipped with a language for defining knowledge elicitation strategies.

6. Knowledge-based integration of representation formalisms

The previous section described how CUE supports model construction and model instantiation. This section presents an approach to model compilation where an explicit ontology is used to select appropriate representation formalisms and reasoning techniques. The approach is based on two principles. Firstly, an attempt is made to use existing problem solvers when possible.† Secondly, it is assumed that in many cases it will not be possible to find a single problem solver that is appropriate for implementing the entire reasoning process.

The basic idea is that different parts of the reasoning process can be implemented by different problem solvers. These problem solvers should be selected in such a way that they are adequate for the part of the reasoning process for which they are responsible. Problem solvers that cooperate to solve a problem must be able to communicate. This communication is complicated by the use of existing problem solvers which may use different representation formalisms. To achieve cooperative problem solving in this situation requires the specification of what the expressions in the formalism of one problem solver mean in the formalisms used by other problem

† We use the term ''problem solver'' for a combination of a reasoning technique and a representation formalism.

solvers. That is, the representation formalisms of the problem solvers must be integrated. This section presents a possible approach for realizing such an integration.

Section 6.1 explains why it is in general not possible to find a problem solver that is appropriate for implementing the entire reasoning process which is modelled in the knowledge model, thus illustrating the need for using multiple problem solvers and representation formalisms. Section 6.2 discusses some problems with hybrid integration, the way in which representation formalisms are usually integrated. Section 6.3 presents an alternative way of integrating problem solvers which is called knowledge-based integration. Section 6.4.1 describes a prototype implementation of an architecture that supports knowledge-based integration in CUE, and in Section 6.4.2 and Section 6.4.3 the impact of knowledge-based integration on knowledge engineering and problem solving is discussed. In Section 6.5 the present proposal is compared with other recent proposals in the literature.

6.1. THE NEED FOR MULTIPLE REPRESENTATIONS

It is widely acknowledged in the artificial intelligence community that there are different types of knowledge. For example, a number of researchers have identified dimensions according to which knowledge can be classified [e.g. deep knowledge vs. shallow knowledge (Steels, 1985), causal knowledge vs. heuristic knowledge (Console & Torasso, 1988; Simmons, 1992), knowledge of structure and behaviour vs. functional knowledge (Abu-Hanna, Benjamins & Jansweijer, 1991)].

For reasoning with these different types of knowledge, a large number of problem solvers have been developed which use different knowledge representation formalisms and have different inferential capacities. The reason for these differences is that problem solvers should be sufficiently expressive to represent all the relevant knowledge in a natural way and have the inferential power to derive all the interesting implications of the represented knowledge in an efficient way. As argued by Levesque and Brachman (1985) these two requirements are antagonistic. Therefore every problem solver must make a trade-off between expressiveness and inferential power.

The problem solvers that have been developed can be characterized by the way they trade representational power for inferential power. The restrictions that have been put on the expressiveness to keep inferencing in these languages tractable can be categorized into three classes as follows.

- Firstly, there are representation languages that maintain tractability by putting syntactic restrictions on the expressions in the language. For example, many production rule interpreters can only handle facts in the form of triples. Another example is Prolog, which is based on first-order logic, but does not allow negation or disjunction in the conclusions.†
- A second group of representation languages derive their inferential power from epistemological assumptions about the structure of knowledge. Frame-based representations are typical examples of this category; these systems assume that

† We are only referring to Prolog as a representation formalism here. Of course, Prolog can be used as a programming language for implementing interpreters for other representation languages, which make the trade-off between representational power and inferential power in another way.

knowledge is typically organized as structured objects that are arranged in subsumption hierarchies, and they provide efficient reasoning schemes for such structures. The main difference between this category and the previous one is that here inferential power is achieved by adding specific forms of expressiveness, whereas in the previous category inferential power is achieved by reducing the expressiveness.

- Unlike representation languages in the first two categories, representation schemes in the third category explicitly delimit the range of domains for which they can be used. Representation languages in this category maintain inferential power by making ontological assumptions about the domains and the task that they will be used for. A typical example of this approach is GDE (de Kleer & Williams, 1987), a system for model-based diagnosis, which requires that the devices that are to be diagnosed can be represented in terms of interconnected components.

Besides computational tractability, a second requirement for knowledge representation formalisms for KBSs is *epistemological adequacy*: a representation formalism must be able to reflect all the distinctions that are important for performing a particular task in a particular domain, while it should not force the knowledge engineer or the domain expert to make additional, irrelevant distinctions. The epistemological adequacy of a representation formalism depends on the type of knowledge that needs to be represented. For instance, it has often been noticed that the production rule formalism is well-suited for representing heuristic, associational knowledge, but is ill-suited for causal models (e.g. Simmons, 1993).

Since solving real world problems often involves different types of knowledge, the requirements of epistemological and computational adequacy imply that a KBS that is to solve these problems must be able to use multiple representations and reasoning techniques. The use of multiple problem solvers poses a potential problem: how are the different problem solvers to be integrated? It was mentioned that the representation formalisms that the problem solvers use can have a different syntax, can be based on different epistemological assumptions, and can make different ontological commitments. To make such diverse problem solvers cooperate clarification is required as to how expressions in one formalism map onto expressions in another formalism. In existing systems that use multiple representations, integration is usually realized by predefined syntactical mappings. The next section presents some problems with this kind of integration.

6.2. HYBRID KNOWLEDGE REPRESENTATION

For different types of knowledge, the trade-off between expressive power and computational power should be made in a different way. This observation has initiated the development of a number of reasoning systems that use multiple representations and multiple inference engines [e.g. KEE (Fikes & Kehler, 1985), LOOM (MacGregor, 1991), KRYPTON (Brachman, Fikes & Levesque, 1985) and CYCL (Lenat & Guha, 1990)]. In these so-called *hybrid architectures* the different components are tightly connected: a problem solver can send queries to other problem solvers and use their result for its own reasoning.

In hybrid architectures the problem solvers are typically specialized for particular

```
(defframe ⟨identifier⟩ (⟨type⟩)
  (⟨slot-name⟩ ⟨value⟩)
  . . .)
                ↧
⟨type⟩ (⟨identifier⟩) ∧
⟨slot-name⟩ (⟨identifier⟩, ⟨value⟩) ∧
. . .
```

FIGURE 34. A possible mapping between different representations in a hybrid system. In this mapping, the type of the frame is mapped onto a unary predicate and the slots of the frames are mapped onto binary predicates.

types of (sub-)problems. In LOOM for example, one reasoning engine, which uses a semantic network representation, is responsible for terminological reasoning, while another problem solver, which uses a logical representation, is responsible for assertional reasoning. Another combination of problem solvers which is often found in hybrid architectures is the use of a first-order theorem prover for deductive reasoning and a frame-based problem solver for default reasoning.

In these hybrid systems the integration issue arises. When one of the problem solvers wants to invoke another problem solver it must know how to formulate the query for the other problem solver. In hybrid architectures this problem is solved by specifying mappings between the different representation formalisms. Of course, these mappings can only be partial: the differences in the expressive power between the formalisms is the main reason for having the hybrid architectures. The partial mappings specify the interface between the representation formalisms. For example, a partial mapping between a frame-based representation and a logical representation could specify that frames are mapped onto unary predicates and slots and slot-values onto binary predicates. Figure 34 gives an example of such a mapping for the case of frames and predicate logic.

As can be seen in Figure 34, the integration is only based on the syntactical structure of the different formalisms: whenever something is represented as a frame slot in the frame language, it will be interpreted as a binary predicate in predicate logic. In hybrid systems, the integration is realized by mappings between syntactical structures. This is necessarily so, because in these systems the mappings are defined when the hybrid tool is built. At that moment, the syntactical structures are the only invariants available on which the mappings can be based.

A disadvantage of integration as realized in hybrid systems is that the fixed mappings constrain the ways in which the representations can be used. When it is decided to represent a particular piece of knowledge in some way in one of the representation formalisms, this puts constraints on the ways in which knowledge can be represented in the other formalisms. Therefore, it is not always possible to exploit the full power of all the constituting formalisms. Further, sometimes the hybrid solution is not feasible because there is no obvious way in which the syntactical structures of one formalism should be mapped onto the syntactical structures of another formalism.

For these reasons, hybrid representation is not the optimal solution for the present purpose: representing instantiated knowledge models of arbitrary expressiveness. The next section presents an alternative, more flexible approach for integrating multiple problem solvers.

6.3. KNOWLEDGE-BASED INTEGRATION

The problem with hybrid representations is that they integrate formalisms based on the syntactical *form* of the expressions in the constituting languages. Therefore, it is not always possible to represent a piece of knowledge in the form that is most appropriate for that piece of knowledge in each of the formalisms.

Instead of a syntactical mapping, it is also possible to integrate formalisms based on the *content* of the knowledge. This is what we call knowledge-based integration. Basically, the idea is the following. When a knowledge engineer starts a project, one of the first tasks is to construct a knowledge-level model of the domain knowledge that is required to perform the application task. This model must be formulated in a language that is formal but which may have an unlimited expressiveness, since it will not be used for reasoning. One component of this knowledge-level model is the *application ontology,* which makes the underlying structure of the domain knowledge explicit. When the knowledge model is completed, the knowledge engineer selects a number of problem solvers for implementing the system. Every problem solver has an associated *representational meta-model.* This is a model that specifies what can be represented in a particular formalism, but which abstracts from the syntactical details of the representation. The integration of the different representation formalisms of the problem solvers is then realized by specifying *mappings* between the application ontology and the representational meta-models. Figure 35 shows the difference between knowledge-based integration and hybrid integration.

Application ontology. The application ontology is a specification of the domain knowledge that is needed to perform a particular task in a particular domain. The following logical sentence is a typical example of an expression that would be part of an application ontology in a medical domain. It states that a finding is a tuple that

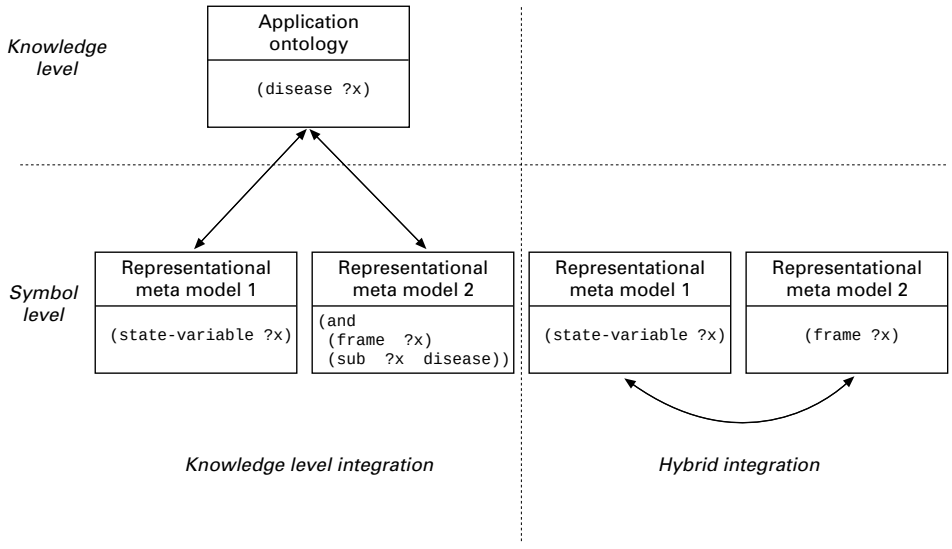

FIGURE 35. The difference between knowledge-based integration and hybrid integration is that the former is based on a knowledge-level application-specific mapping, whereas the latter is based on a symbol-level syntactic mapping.

consists of an observable, an operator and an element of the value set of the observable.

$$\forall X, Y, Z\ \mathsf{finding}(X, Y, Z) \rightarrow \\ [\mathsf{observable}(X) \wedge \mathsf{operator}(Y) \wedge Z \in \mathsf{value\_set}(X)]$$

Representational models. The representational meta models are abstract descriptions of the types of expressions that are allowed in a knowledge representation formalism. They are formulated in the same language as the application ontology. The representational meta-models of problem solvers can vary from coarse-grained descriptions of conditions and actions in the case of production rule interpreters to fine-grained ontological models of what can be represented in systems like GDE. For example, the following expression, which states that a condition consists of a set of attribute expressions, could be part of the representational meta-model of a production rule interpreter, where the rules have condition parts and action parts.

$$\forall X,Y\ [\mathsf{condition}(X) \wedge Y \in X] \rightarrow \mathsf{attribute\_expression}(Y)$$

Mapping. The mappings between the application ontology and the representational meta-models specify how expressions from the language defined by the application ontology can be translated into the representation formalisms and vice versa. This serves two goals: (i) it specifies how the knowledge-level model can be implemented on a computational architecture in the design phase for the application and (ii) the mapping specifies the meaning of the expressions of the representation formalism in terms of the application ontology. As there is a mapping for each of the participating problem solvers, the application ontology specifies a language that is understood by all. Therefore, it can be used for communication between problem solvers. Mappings which are only used for the first goal are called static mappings. They are only used once, during the construction of the system. Mappings which are used for communication between problem solvers, are called dynamic. These mappings are used at run time, to translate output from one problem solver into input for another problem solver.

As an example of the mapping between an application ontology and a representational meta-model, consider the following mapping rules:

$$\mathsf{finding}(X, Y, Z) \mapsto \mathsf{attribute\_expression}(\langle X, Y, Z \rangle)$$
$$[\mathsf{disease}(X) \wedge \mathsf{qualitative\_probability}(X, Y)] \mapsto \mathsf{attribute\_expression}(\langle X, =, Y \rangle)$$

Conceptually, these mappings are quite simple. For instance, the first rule expresses that findings, as defined in the application ontology, are represented as attribute expressions in the production rule formalism. However, in this simple example there are some technical complications that the mapping mechanism should account for. Whereas `finding` is a ternary predicate in the application ontology, `attribute_expression` is a unary meta-predicate in the representational meta-model. In the example, the mismatch is purely notational: in the representational meta-model `attribute_expression` is defined to hold for three-placed tuples that have an operator as their second element. However, the knowledge engineer should be aware of conceptual incompatibilities, in which case the selected problem solver is not epistemologically adequate.

The mapping mechanism must be able to perform the mappings in both

directions. In the example, this is possible because the qualitative probability relation is defined to have a disease as its first argument. Therefore, the type of the first element of the attribute expression tuple can be used to decide on the corresponding application ontology expression: if it is an observable, the corresponding expression is a finding, if it is a disease, the corresponding expression is a qualitative probability.

Although the mapping relations will usually be more complicated than the ones shown here, they should remain relatively simple. When it is not possible to define simple mappings, this could indicate that the expressiveness of the problem solver is not sufficient for expressing the distinctions made in the application ontology. For example, if the attribute expressions in the representational meta-model only allow the $=$ operator while in the findings in the application ontology also the $<$ and $>$ operators are used, complicated mapping rules would be required. The complexity of the mapping relation can therefore be viewed as an measure for the suitability of the problem solver: if the mappings are simpler, the problem solver is more appropriate.

As argued in Section 6.1 it is often impossible to find a problem solver that is able to represent the full spectrum of knowledge types that is specified in the application ontology. In such cases, the task that the application must perform is broken up into subtasks, each of which is associated with a problem solver. Problem solvers must be selected so that the knowledge that is needed to perform the particular sub-task can be adequately represented in the problem solvers' representation.

To summarize, the difference between hybrid integration and knowledge-based integration is that in the former the integration is realized directly, while in the latter the integration is realized through an intermediate knowledge-level model: the application ontology.

6.4. APPLYING KNOWLEDGE-BASED INTEGRATION

Using knowledge-based integration has important consequences for the practice of knowledge engineering and thus for the required functionality of AI toolkits. Since knowledge-based integration is based on the contents of the application ontology, the integration can only be realized *after* knowledge acquisition. Therefore, the decision on how to integrate the selected problem solvers must be taken by knowledge engineers when they develop an application. Knowledge engineering methodologies should recognize this activity as an integral part of the knowledge engineering process and provide methods and tools to support it. Section 6.4.1 describes a prototype of a toolkit that supports knowledge-based integration. In Section 6.4.2 it is illustrated how this system can be used to develop an application and Section 6.4.3 illustrates the impact of knowledge-based integration on the problem-solving process.

6.4.1. Knowledge-based integration in CUE

Section 5 presented CUE's knowledge acquisition tools: QUITE, QUOTE and QUAKE. The output of these tools is a non-executable knowledge-level description of the domain knowledge needed for an application. This section presents CUE's facility for developing design models. A first prototype of this module has been implemented.

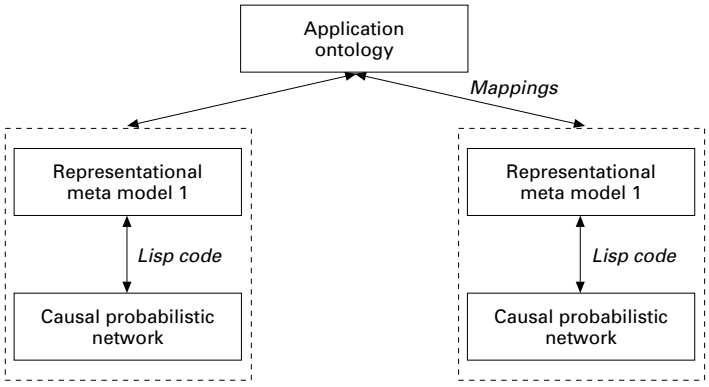

FIGURE 36. Mappings and translations in knowledge-based integration.

CUE is an open architecture that allows arbitrary problem solvers to be plugged-in. This requires two steps: (i) the representational meta-model of the problem solver must be constructed and (ii) it must be specified how this model is related to the internal representation of the problem solver. In CUE, the representational meta-models are also specified in Ontolingua.† The links between the meta-models and the internal representations are realized by problem solver specific translators, written directly in Lisp. An example of the architecture of an application that is constructed with CUE is depicted in Figure 36.

Whereas the mappings between the application ontology and the representational meta-models must be specified for every application, the Lisp code for translation between the meta-models and the internal representations of the problem solvers needs to be specified only once. When this is done, the problem solvers and the associated representational meta-models and translation code can be put into a library for reuse. Thus, while the mapping relation must be specified by the knowledge engineer, the translation code will be written by the developers of CUE's problem solver library.

6.4.2. An example

The use of CUE will be illustrated with some fragments from a scenario which is based on an exercise to reconstruct parts of the FREECALL system (Post, Koster, Zocca & Sramek, 1993) in CUE. The exercise was intended to test the idea of knowledge-based integration, and not to develop a realistic system. Therefore, some of the design decisions may seem odd from an engineering perspective.

FREECALL is a KBS that supports ambulance dispatchers in their decision whether to send an ambulance after an emergency call. In the exercise, we concentrated on two sub-tasks of the system: (i) the generation of a set of initial hypotheses, and (ii) the assessment of the likelihood of these hypotheses. In the example only the use of dynamic mappings will be illustrated. Section 7 will present some examples of static mappings.

† Although Ontolingua is used for both the application ontology and the representational meta-models, we do not entirely agree with the Ontolingua philosophy. Section 6.5 discusses the exact relation between the present work and the work of Gruber.

The application ontology of FREECALL contains definitions of findings (as in Section 6.3), diseases, and the way they are related. Because of the large number of potential hypotheses—a computational consideration—it is decided to use production rules to generate the initial hypotheses set. The mapping between the application ontology and the representational meta-model of the production rule interpreter is described in Section 6.3. For the second sub-task IDEAL is selected from the problem solver library. IDEAL (Srinivas & Breese, 1990) is a system for evaluating causal probabilistic networks and influence diagrams. The representational meta-model of IDEAL defines the class `state_variable` and the relation `influences`.†

$$\forall X, Y\ \mathsf{influences}(X, Y) \rightarrow [\mathsf{state\_variable}(X) \wedge \mathsf{state\_variable}(Y)]$$

The relation `state_expression` is defined as follows.

$$\forall X, Y\ \mathsf{state\_expression}(X, Y) \leftrightarrow [\mathsf{state\_variable}(X) \wedge Y \in \mathsf{value\_set}(X)]$$

Further, the representational meta-model specifies that every state expression S_i has a number of associated conditions $C_1, \ldots, C_n$. A condition C_j is a set of state expressions $S_1, \ldots, S_n$ with a state expression $S_k \in C_j$ for every state variable that influences the state variable of S_i. For every condition C_j that is associated with S_i there is a conditional probability $P(S_i \mid C_j)$ so that $0 \leq P(S_i \mid C_j) \leq 1$.

For IDEAL, the mapping between the application ontology and the representational meta-model is more complicated than for the production rules. It is decided that observables are represented as state variables and findings as state expressions about these variables. Diseases are represented as state variables too, with two admissible values: present and absent. Alternatively, the diseases could be represented as admissible values on a state variable `diagnosis`, but this would make it impossible to hypothesize multiple diseases. Furthermore, it would make it impossible to use probabilities as described in the medical literature—an epistemological consideration.

In the representational meta-model, state expressions are represented as binary relations between state variables and values. There is no explicit mentioning of an operator, it is assumed that only the equality operator is used. Therefore, IDEAL can only be used when the findings only use the equality operator. This is another example of an epistemological consideration.

The following expression is an example of the (dynamic) mapping between findings and state expressions. It states that if a particular finding holds, this means that the probability of this finding is 1.0.

$$\mathsf{finding}(X, Y, Z) \mapsto P(\mathsf{state\_expression}(X, Z)) = 1.0$$

6.4.3. Running a CUE application

The impact of knowledge-based integration on the problem solving process can be illustrated with a session with the above described version of FREECALL. In the example, the caller is a man whose 28 year old son complains about lasting chest pain. The trace in Figure 37 shows example mappings and translations at the stage

† We use the logical notation instead of the Ontolingua notation which is used in CUE. The logical notation is more concise and probably more familiar.

System's action	comment
(1) initial-data: finding(chest_pain, =, present) initial-data: finding(sustained_pain, =, yes) initial-data: finding(age, =, 28)	The patient data are entered as findings, as defined in the application ontology. The label initial-data indicates the role of the findings in the reasoning process.
(2) *mapping* attribute_expression (⟨chest_pain, =, present⟩) attribute_expression(⟨sustained_pain, =, yes)⟩	Because the hypotheses generation sub-task is assigned to the production rule interpreter, the findings are rewritten in terms of the representational meta-model of the production rule interpreter

(3) *Lisp translation*
`IF (and (=chest-pain present)`
`(=sustained-pain yes))`
`THEN (=angina-pectoris possible)`
	and then further translated into the private representation of the production rule interpreter, so that it can be matched against rules such as the one shown here.

(4) *Lisp translation* attribute_expression(⟨angina_pectoris, =, possible⟩) attribute_expression(⟨hyperventilation, =, possible⟩) attribute_expression(⟨infarction, =, probable⟩)	The production rule system generates three hypotheses, together with a qualitative assessment of their likelihood. This output is translated back into the representational meta-model language.
(5) *mapping* hypothesis: disease(angina_pectoris) qualitative_probability(angina_pectoris, possible) hypothesis: disease(hyperventilation) qualitative_probability(hyperventilation, possible) hypothesis: disease(infarction) qualitative_probability(infarction, probable)	Finally, the output is rewritten in terms of the application ontology. The generated diseases are assigned the role of hypotheses (this is control information). This completes the hypotheses generation sub-task in the FREECALL system.

FIGURE 37. Trace of mappings and translations for hypothesis generation using a production rule interpreter.

where FREECALL generates the initial hypotheses. Note that whereas in step 2 in this figure the findings are rewritten as attribute expressions, in step 5 attribute expressions are rewritten as qualitative probability assessments.

The second sub-task in the FREECALL system is a quantitative assessment of the probabilities of the disease in the differential—the set of hypotheses generated with the production rule interpreter. Hence, an influence diagram is selected which contains nodes for all of the diseases in the differential and for the observables of the known findings (the initial data). The mapping rules specified in the previous section are used to set the probabilities of the state expressions that represent the known findings to 1.0 and then IDEAL is invoked. A trace of the mappings and translations that are required for this form of hypothesis discrimination is shown in Figure 38. From the selected influence diagram, FREECALL derives that hyperventilation is by far the most likely diagnosis. Note that in step 7, the expressions in terms of IDEAL's representational meta-model are translated directly into Lisp function

System's action	Comment
(6) *mapping* P(state_expression(chest_pain, yes)) = 1.0 P(state_expression(age, 25–29)) = 1.0	The findings and the hypothesized diseases are mapped onto state expressions in the influence diagram. The probabilities of the state expressions that represent findings are fixed to 1.0.
(7) *Lisp translation* `(setf (prob-of '((#n(chest-pain) #c(yes chest-pain)))) 1)`	Then, the probability assignments are translated into Lisp function calls that set the values of structures in the internal representations of IDEAL.
(8) *Lisp translation* P(state_expression(angina_pectoris, present)) = 0.03 P(state_expression(infarction, present)) = 0.04 P(state_expression(hyperventilation, present)) = 0.27	Next, the influence diagram is evaluated and the resulting probabilities are translated back into the representational meta-model terminology.
(9) *mapping* quantitative_probability(hyperventilation, 0.27) quantitative_probability(angina_pectoris, 0.03) quantitative_probability(infarction, 0.04)	Finally, the expressions are translated back into the application ontology language and presented to the user of the system.

FIGURE 38. Trace of mappings and translations for hypothesis discrimination using an IDEAL influence diagram.

calls. The reason for this is that IDEAL has no declarative knowledge representation. This is another reason why representational meta-models are necessary for knowledge-based integration.

The traces in Figure 37 and Figure 38 illustrate how statements in the application ontology can be translated into specific representation formalisms for executing an inference method. The mappings and translations ensure that the results are meaningful in terms of the application ontology.

6.5. DISCUSSION

Although the term ''knowledge level'', was used occasionally, this section addresses a ''symbol level'' issue: how to integrate knowledge representation formalisms, or more specifically: what do expressions in one formalism mean in another formalism. This is just one aspect of the larger issue of having multiple problem solvers, or agents, cooperate to solve a problem. We have—intentionally—ignored control issues. This scoping decision was made because we believe that the problem of integrating representation formalisms can be resolved independently from the control problem. This has the advantage that such a solution can be used within a wide range of control architectures. For example, in the scenario above the decision to invoke IDEAL could be taken in a data-driven way, as in blackboard systems, or in a goal-driven way, as in task-oriented architectures.

The main message of is that in cases where it is necessary to use multiple representation formalisms, the application ontology can be used to integrate these

formalisms in such a way that the strengths of the different formalisms can be exploited maximally.

As already mentioned, the work presented here is closely related to the work on Ontolingua (Gruber, 1993). CUE uses the Ontolingua language both for the application ontology and the representational meta-models. Besides a language for specifying ontologies, Ontolingua is also a computer program that translates the ontologies into the representation formalisms of a number of problem solvers. In Ontolingua, the translation of ontologies to representation formalism does not require additional knowledge. The hypothesis that underlies the Ontolingua program is that it is possible to specify once and for all how expressions in the knowledge-level language are to be represented in the target representations. Therefore, the integration as realized through Ontolingua is essentially hybrid: when something is represented in a particular way in one representation it is predetermined how that knowledge will be represented in another representation. In contrast, in knowledge-based integration the translation is viewed as a knowledge intensive activity, which must be performed by the knowledge engineer. To facilitate this activity, the translation process is divided in a knowledge intensive part, the mapping operation, and an automatic part, the translation into the representations.

We have not been very specific about the exact nature of the mapping relation. As in hybrid integration, the mappings may be partial: they connect the representational meta-models only with those parts of the application ontology that specify the knowledge needed to perform the particular sub-task assigned to the problem solver. In general, we can formulate one hard constraint and one soft constraint on the mappings. The hard constraint is that the mappings must be bidirectional: it must be possible to go from the application ontology expression to the representational meta-model expression and it must be possible to go back from the representational meta-model expression to the application ontology expression. If this is not possible, the problem solvers lose the ability to communicate. The soft constraint is that the mappings are to remain simple. As was mentioned in Section 6.3, the complexity of the mapping relation is inversely related to the suitability of the selected problem solver.

In Section 6.3 it was mentioned that there are two types of mappings; static mappings, which are used during application development, and dynamic mappings, which are used during problem solving. In the examples so far all the mappings were dynamic mappings. Section 7 will give some examples of the use of static mappings during KBS development.

7. Treating acute radiation syndrome: a case study

In this section we illustrate the use of explicit ontologies to develop a system that supports the treatment of acute radiation syndrome (ARS): a collection of injuries caused by exposure to high dosages of irradiation. ARS is a rare disorder; world wide there are about 900 known cases. The expertise for treating ARS is scarce and, because of the increased safety of nuclear power plants, it is decreasing. When such expertise is needed, however, in cases of nuclear accidents or nuclear attacks, it is likely to be needed immediately, and on a large scale.

For this reason, researchers at the University of Ulm, which is one of the centres

of expertise for managing acute radiation syndrome, have decided to develop a knowledge-based decision-support system (Kindler, Densow & Fliedner, 1993). A prototype for such a system was implemented using M-KAT (Lanzola & Stafanelli, 1992). This section describes a reimplementation of the system, using the CUE tools.

7.1. SYNOPSIS OF THE ARS DOMAIN

A large dose of radiation causes depletion of cells, particularly in tissues where the cell population is normally renewed by continuous cell division and maturation. Organ systems that are affected by radiation include the haemopoietic system, the reproductive organs, the gastrointestinal tract, the skin, and the central nervous system. The stem cells of the bone marrow, which are responsible for the production of blood cells, are particularly susceptible to the harmful effects of radiation. One type of blood cell, the leucocytes (white blood cells), play a fundamental role in the immune system. One task of the stem cells is to ensure that the number of leucocytes is kept at a certain level. There are two main types of leucocytes; granulocytes, which are produced in the bone marrow, and lymphocytes, which are produced in the lymphogeneous organs, including the lymph glands and the thymus.

The main aim of the treatment of ARS is to control the development of immunodeficiency. Immunodeficiency develops when the number of surviving stem cells after irradiation is too low to produce the necessary number of granulocytes, and, to a lesser extent, lymphocytes. In order to prevent the development of immunodeficiency, which can be lethal, a bone marrow transplantation (BMT) must be considered. However, BMT might have severe side effects, such as graft-versus-host disease (GVHD). In addition, the injuries to other organ systems might be so severe that the patient would not survive anyway, in which case a bone marrow transplantation should be avoided. When a considerable number of stem cells survives the radiation exposure, growth factor therapy may be considered as an alternative. In this therapy, the patient is treated with *growth inducers,* proteins that stimulate the growth and reproduction of stem cells.

A first target of ARS treatment is to establish the severeness of the radiation injury, expressed in terms of the estimated damage to four organ systems: the haemopoietic system (of which the stem cells of the bone marrow are a part), the skin, the gastrointestinal tract and the central nervous system. As a result of the exposure to radiation, these systems develop time-dependent patterns of signs and symptoms. These patterns must be interpreted to assess the severity of the lesions to each of the four systems. The lesions to each of the systems are expressed in terms of severity gradients—labels for qualitatively different severities. Based on these gradings appropriate therapeutic action can be undertaken.

7.2. MODELLING THE TASK

The aim of ARS management is symptomatic treatment—the radiation itself is irreversible. The medical expert must make a decision as to whether a particular action must be undertaken to control processes that occur in the patient. To make such a decision, the medical expert attempts to establish the severity of each of the four lesions that might be the result of the radiation. In the task model this situation

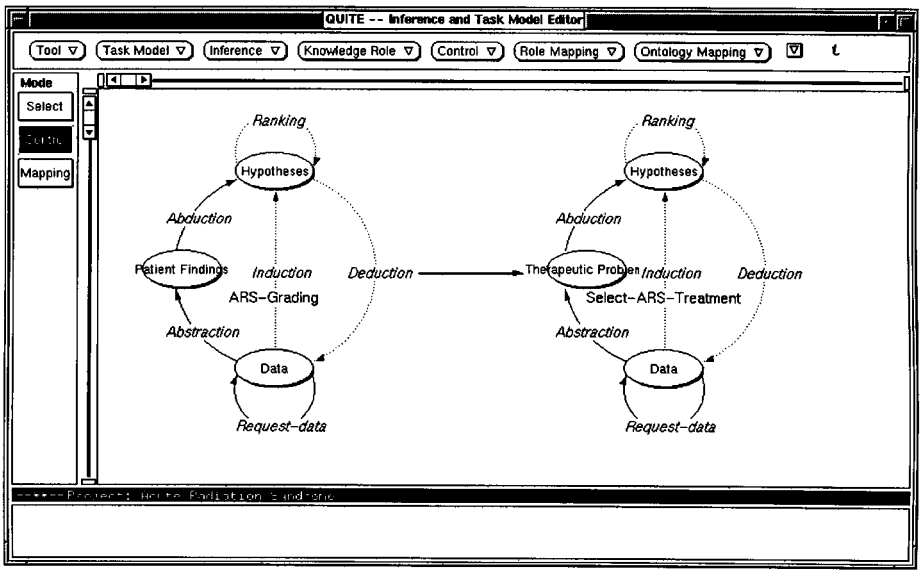

FIGURE 39. The task model for the ARS application.

is modelled by means of a diagnostic STModel, grading the severity of the syndrome, and a therapeutic STModel, selecting the appropriate action.

The system receives its input from a standardized medical record which has been developed for the structured documentation of ARS cases (Baranov, Densow, Fliedner & Kindler, 1994). This record contains the data that might be relevant for ARS treatment. Since all the relevant data are entered before the KBS is invoked, the system is not required to deduce expectancies and to request new data. Thus, only the abstraction step and the abduction step of the diagnostic cycle have to be performed by the system. For the same reason, the system only performs abstraction and abduction in the therapy planning sub-task.

Figure 39 shows a first version of the task model, constructed with QUITE. The knowledge engineer has created a diagnosis generic-task instance (ARS-grading) and a therapy-planning generic-task instance (Select-ARS-treatment). These are connected by a control link which indicates that grading precedes treatment selection. For the moment, this is all that can be specified in the task model. The specification of the role-to-role mappings requires a better understanding of the nature of the domain knowledge. At this stage of the modelling process, the task model only provides a rough description of the reasoning process. This model is sufficient to initiate construction of the application ontology.

7.3. BUILDING THE APPLICATION ONTOLOGY

The GAMES-II project has developed a library of ontological theories that can be used to develop application ontologies. This library is indexed by medical sub-domains and reasoning methods. Ontological modelling therefore starts by giving a rough indication of the medical sub-domain and the possible reasoning methods.

Treatment of acute radiation syndrome involves at least four medical sub-fields:

Guidelines for deciding on for which concepts to look:

Guideline A.1: *Determine the concepts that play the primary roles in the reasoning process.* The primary roles are the roles that always recur in the STModels for a particular generic task, independently of the particular methods that are used. For the diagnostic part of the task model these are diagnostic hypotheses, patient findings and data, and for the therapeutic part therapeutic hypotheses, therapeutic problems and data.

Guideline A.2: *Determine the ontological feature of the concepts that are used to make the basic inferences in the reasoning process.* These are abstraction, abduction, ranking, deduction, and induction. It is usually better to search first for the concepts that play the primary roles (guideline A.1) because the terminology for these concepts is more standardized than the terminology for the concepts used for inferencing.

FIGURE 40. Guidelines for deciding on the order in which the library should be searched for concepts.

haemotology, dermatology, neurology and gastroenterology. Currently, the library contains no extensions that are specific to these sub-fields. Therefore, the domain-specificity indexes cannot be used directly to find the appropriate concepts.† It is also not possible to decide at this moment which method-specific extensions are needed; the suitability of methods often depends on the ontological structure of the knowledge in the application domain.

For using the library in this situation, a number of guidelines have been developed. A first set of guidelines, presented in Figure 40, can be used to decide on the order in which a knowledge engineer should search for concepts in the library. A second set of guidelines is intended for deciding on how to look for a particular concept in the library. These guidelines are presented in Figure 41. A third set of guidelines, presented in Figure 42, can be used for deciding on whether a particular concept is suitable for the present purpose.

The next sections will describe how these guidelines were used in the ontological modelling process for each of the two generic task instances. The results of this process are summarized in Figure 45.

7.3.1. Ontology for ARS-grading

The order in which the ontology for ARS-grading is constructed is based on guidelines A.1 and A.2. First we select or construct the concepts that play the knowledge roles, and then we deal with the concepts that are used for making the inferences.

Diagnostic hypotheses. As we have seen in the domain description in Section 7.1, the hypotheses role is mapped onto alternative gradings of the four syndromes that form ARS: the haemopoietic syndrome, the gastrointestinal syndrome, the skin syndrome and the central-nervous-system syndrome. We first concentrate on modelling syndromes, and then on their gradings.

Following guideline B.1, the library is searched for a concept with the name ''syndrome''. This concept is found in the theory syndrome, which is selected from

† We use the term concept in the most general sense: classes, relations and functions are all concepts.

Guidelines for deciding on how to look for a concept:

Guideline B.1: *Ask the domain expert to suggest a name of the concept that is used for the particular role in the domain and search the library for a concept with a similar name. If the search succeeds, go to guideline B.2 Otherwise go to guideline B.5.* The idea here is that if the appropriate concept is somewhere in the library, it can be found by terminology matching. This is more likely to succeed for concepts that are used for knowledge roles than for concepts that are used for inferences, because the terminology for the former is more standardized.

Guideline B.2: *Check if the definition found is suitable for the current purpose. If the concept is appropriate, include it in the application ontology. Otherwise, go to guideline B.3.* When a concept with the right name is found, this does not guarantee that the concept definition is appropriate, so this needs to be checked.

Guideline B.3: *Find out if the concept can be specialized to a suitable sub-concept. If this is possible, add the specialized concept to the application ontology. Otherwise, go to guideline B.4.* When a library concept is inappropriate because it is too general, it can be made appropriate by adding specific details. This is done by introducing an application specific sub-concept of the library concept.

Guideline B.4: *If the concept found is not suitable and it can also not be specialized to a suitable sub-concept, find out if it can be modified to become suitable. If this is possible, copy the concept to an application-specific part of the application ontology, modify it and rename it to avoid name conflicts. Otherwise, go to guideline B.5.* If the concept found by terminology matching is not appropriate, it can still be useful. During the discussion with the domain expert it becomes clear which aspects of the concepts are appropriate and which are inappropriate. This information can be used to define a more suitable version of the concept in the application ontology.

Guideline B.5: *If no suitable concept can be found or constructed using guidelines B.1 to B.4, try to find a very general concept in the core library and try to determine in discussion with the domain expert whether this concept can be specialized or modified to arrive at an appropriate concept.* When a general concept is selected, it is almost always necessary to specialize it.

FIGURE 41. Guidelines for deciding on how to search in the library for specific concepts.

Guidelines for deciding on the suitability of a particular concept

In general, a concept is suitable if it makes the distinctions that are necessary for the present purpose, and no other distinctions. This can be operationalized by means of the following guidelines:

Guideline C.1: *Decide whether the concept is sufficiently general to cover the piece of knowledge that will be modelled using the concept.* If a concept is not sufficiently general, knowledge elicitation will become problematic because some pieces of knowledge that are used in the reasoning process cannot be modelled.

Guideline C.2: *Decide whether the concept is sufficiently specific to only cover the pieces of knowledge that will be modelled using the concept.* If a concept is not sufficiently specific, it is not possible to define restrictive mappings between the application ontology and the task model: too many concepts will be allowed to play too many roles.

Guideline C.3: *Decide whether the name of the concept is a meaningful term in the application domain.* If the name of the concept is not sufficiently domain specific, it cannot support knowledge acquisition in domain-specific terminology.

FIGURE 42. Guidelines for deciding whether a particular concept is appropriate for the present purpose.

the library. According to guideline B.2 it must now be checked as to whether it is suitable for the current purpose. The theory syndrome defines syndromes as collections of findings which cooccur but for which there is no known direct causal connection. Syndromes are modelled as a sub-class of disorder. According to this definition, ARS is the only real syndrome in the domain; the component syndromes of ARS are processes about which the causal mechanisms are reasonably well understood. Thus, for the ''syndromes'' that constitute ARS the definition is inappropriate.

Following guidelines B.3 and B.4, it is investigated whether the definition of `syndrome` can be specialized or modified to make it suitable. It is not possible to specialize the concept because a part of the definition of `syndrome` does not hold for the lesions. Specialization can only be used to add attributes to a definition, not to remove attributes. In principle, the concept could be modified to make it appropriate. However, it is decided not to do this because this would involve removing the only aspect of `syndrome` that distinguishes it from its superconcept `disorder`.

The problem is that for the lesions that constitute ARS the term ''syndrome'' is a misnomer. As prescribed by guideline B.5, it is therefore decided to introduce a new specialization of `disorder` which is called `ars-organ-system-lesion`. This concept is added to the theory ars-application, a theory added to the application ontology for storing application-specific definitions.

Modelling different gradings of `args-organ-system-lesion` can be done in two ways: (i) by defining that the possible gradings are sub-types of the lesion, or (ii) by modelling the gradings as expressions *about* the lesion. Examples of both types of modelling can be found in the core part of the library. An example of the first approach is in the theory disease. In this theory, the relation `disease-subtype` is defined, which can be used to model that particular diseases are specializations of other diseases (cf. the examples in Section 5). In the current application, this could be realized by defining a relation `lesion-subtype` between instances of `ars-organ-system-lesion`. This solution is shown in Figure 43(a).

However, the different possible gradings of a lesion to one particular organ system are mutually exclusive: a patient can only have one grading for a particular lesion.

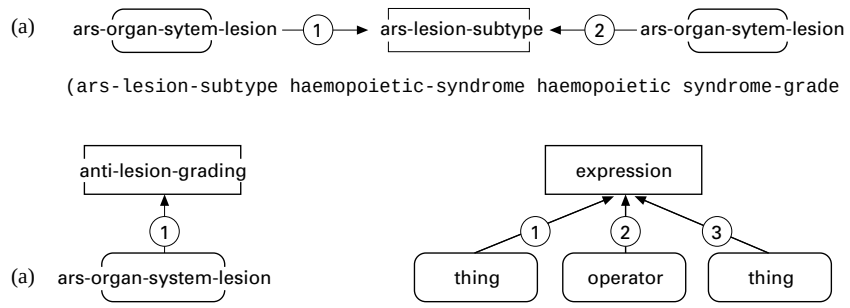

FIGURE 43. Two alternative ways to model lesion gradings. The rectangular boxes represent relations and the rounded boxes represent classes. The numbered arrows represent that the *n*th parameter of the relation pointed at is constrained to be an instance of the lass pointed from.

Characteristics like this are important for knowledge elicitation and for computational design and should therefore be represented in the application ontology. Modelling the gradings this way would therefore require the addition of a number of very specific constraints to the lesion-subtype relation. Although this kind of customization is not unusual, it is often worthwhile in such situations to look for library definitions that better capture the relevant aspects of a concept.

Another general medical concept in the library is finding.† Findings are defined as expressions about patient parameters. Modelling ars-lesion-grading as a specialization of finding would solve the problem mentioned above, because for finding it is already defined that the different findings about one observable are mutually exclusive. There is one complication to make ars-lesion-grading a specialization of finding, ars-organ-system-lesion must be made a specialization of patient-parameter. Because ars-organ-system-lesion was already modelled as a disorder, finding is not suitable because it is too specific (guideline C.1). Following guideline B.5 the library is searched for a more general concept: finding is a specialization of expression, which is defined in the theory expression. ars-lesion-grading is therefore defined as an expression for which the first parameter must be an ars-organ-system-lesion.

To summarize the modelling decisions above: the only real syndrome in the domain is ARS. This syndrome consists of four gradings of disorder which are modelled as ars-lesion-gradings. These gradings play the role of hypotheses in the diagnostic reasoning process. The lesion gradings have as first parameter an ars-organ-system-lesion, which is a specialization of disorder.

Patient findings. For patient findings, the concept finding is used as a first guess for a suitable ontological concept. However, this concept is not entirely appropriate because it is too generic (guideline C.2). In the ARS domain, the patient findings are a specific kind of findings with qualitative value sets. As explained in Section 5 and 6, both knowledge acquisition and computational design are facilitated by being as specific as possible in the application ontology. Following guideline B.3 it is therefore decided to define a specialization of finding in the theory ars-application, named ars-lesion-indication. An ars-lesion-indication is a finding where the observable must be an ars-lesion-indicator. In turn, these lesion indicators are modelled as a sub-class of patient-parameter in ars-application. Further, it is defined that the value of the ars-lesion-indication must be an ars-indicator-value.

Diagnostic data. The KBS receives its inputs in the form of a computerized record. This record contains many different kinds of data whose only shared characteristic is that they contain information about the patient. Using guideline B.1, the library is searched for a concept named ''datum''. Such a concept is not present in the current library. Thus, using guideline B.5 we look for a general concept. Again, the concept finding is selected from the library. According to guideline C.2 finding is not

† Note that the term *finding* is used for both a particular knowledge role in the STModel and for an ontological concept. These are different things, but the term ''finding'' is used for both in medicine. Although this is a continuous source of confusion, we have decided to respect this tradition. For clarity, we will use the term ''finding'' when referring to the ontological concept and the term ''patient finding'' when referring to the inference role.

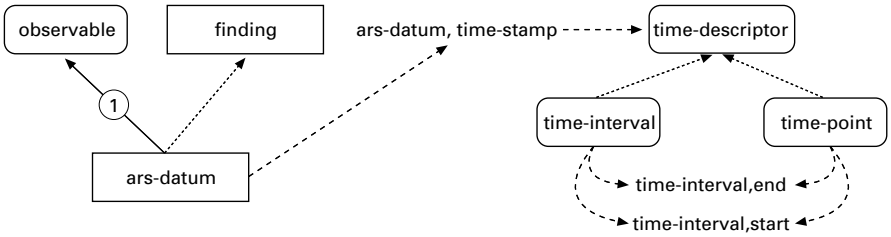

FIGURE 44. `ars-datum` is represented as a specialization of `finding` where the first argument must be an `observable`. Every datum has an associated `time-stamp`, which may be a `time-interval` or a `time-point`. The texts `ars-datum`. `time-stamp`, `time-interval.start` and `time-interval.end` represent functions. The dashed arrows represent the domains and ranges of the functions.

suitable because it is too general: also instantiations of `ars-lesion-indication` are findings. Thus, the concept needs to be specialized. Therefore, the concept `ars-datum` is created and stored in ars-application. When specializing a concept, it must be decided which aspects are shared by the pieces of knowledge that must be covered by the specialization.

One aspect that distinguishes raw data from findings in general is that data are directly observable. This can be modelled by specifying that the first parameter of `ars-datum` must be an `observable`, a sub-class of `patient-parameter` defined in the theory observable.

Another important aspect of data in this domain is their temporal attributes. Currently, the library contains only one simple theory of time which is selected. This theory defines the concepts `time-point` and `time-interval`, but it does not define temporal relations such as ''before'' and ''after''. For determining the appropriate treatment for ARS the time points of data with respect to the time point of the radiation accident are important, but the application will not monitor the patients, so no complex reasoning *about* time is required. Finally, the knowledge engineer defines in ars-application that every datum is associated with a time stamp, which may be a time point or a time interval. `ars-datum` and the related concepts are shown in Figure 44.

To illustrate how data are modelled, consider the part of the medical record which is shown in Table 3. This part of the record is used by doctors to describe erythema in the patient.

TABLE 3
The structure of the record field for erythema

Location	Yes	No	Unknown	Begin	End	Maximum	Degree
Head and neck	×			16.06.1958	22.06.1958	18.06.1958	2
Upper part of body							
Arms							
Lower part of body							
Legs							
Feet							
Oropharyngeal							

In terms of the application ontology, the data in the fields of the record are modelled as tuples such as the following.†

```
(ars-datum head-and-neck-erythema=2)
(ars-datum.time-stamp (ars-datum head-and-neck-erythema=2)
  (time-interval 16.06.1958 22.06.1958))
```

The name of the observable is constructed by concatenating the term ''erythema'' and the term that represents the location of the erythema. The degree-field of the erythema record is represented by the datum-value. Also the epistemic modality of the datum is expressed by means of the datum-value: if the value is a degree or ''no'' it is known, otherwise it is unknown. The ''maximum'' field of the erythema record, which stores the date at which the erythema is at the maximum, is ignored because it is not used in the clinical reasoning process. The begin and end fields are modelled by means of the `ars-datum.time-stamp` attribute.

Diagnostic abduction. After having modelled the concepts that play the primary roles in the diagnostic process, we now turn to the knowledge required to make the inferences (following guideline A.2). In ARS diagnosis, abduction of the hypotheses is a straightforward process—the possible values of the lesion indicators have been chosen in such a way that they can easily be related to the organ-system-lesion gradings. These associations can be modelled by means of direct relations between the lesion indications and the gradings of the organ-system lesions.

The knowledge about the presence of direct associations can be used for exploring the library with the method-specificity index. Using this index, the knowledge engineer retrieves the `manifestation-of` relation from the library, which was also used in some examples in earlier sections. However, this relation is not entirely appropriate because it relates findings to diseases, whereas we are looking for a relation that relates findings to lesion gradings, which are modelled as expressions. Guideline B.3—which suggests specializing the generic concept—is not applicable, because there is a type mismatch between the second parameter of `manifestation-of` (the class `disease`), and the relation `ars-lesion-grading`. Following guideline B.4, `manifestation-of` is therefore copied to ars-application and modified to have an ars-lesion-indication as its first argument and a `ars-lesion-grading` as its second argument. The modified relation is called `ars-manifestation-of`.

In the library version of `manifestation-of`, the tuples of of the relation are qualified with `evoking-strength` and `frequency` attributes. These attributes can be used to characterize the correlation between the diseases and their manifest findings. To determine whether these attributes are useful for `ars-manifestation-of`, the expert is asked if, given a particular finding, some lesion-indications occur more often than others. This turns out not to be the case in the ARS domain—when a lesion has a particular grading all the indications are present and, if not all the indications are present, the evidence is insufficient to make the corresponding diagnosis. This information is recorded in the documentation string that QUOTE associates with the concept.

† For readability, in CUE the complex terms in meta-level expressions are implicitly quoted and labelled by their ontological type.

Diagnostic abstraction. Different kinds of abstraction are used in the ARS domain. On the one hand, there are simple quantitative abstractions that map raw data onto qualitative assessments of the same parameter. On the other hand, there are abstractions that use complex mathematical formulae to compute the value of a lesion indicator from a number of raw data. Because the knowledge used for the abstractions is of a mathematical nature and can be communicated adequately using conventional mathematical notations, it is decided not to model the detailed characteristics of the knowledge used for the abstraction inference in the ontology. That is, we do not model concepts such as sine and cosine which are used in the formulae. Because it *is* important to know which observable data are used to compute the lesion indications, the relation `ars-abstracted-from` is defined in ars-application. This relation can be used to specify which knowledge is needed for the abstractions without specifying how the abstractions are computed. The formulae that describe how the abstractions are computed are only specified in the documentation slots that are associated with every knowledge piece in CUE.

7.3.2. Ontology for select-ARS-treatment

Therapeutic hypotheses. In therapy planning, the most obvious candiates for the hypotheses role are therapies. Using guideline B.1, the concept `therapy` is found in the library theory `therapy`. Because the definition appears suitable for the current applications, this theory is included in the application ontology.

Therapeutic problems. Therapeutic problems are the prime targets of therapy planning. Obviously, in the ARS domain these are in the lesion gradings. However, other information is also needed to decide on the appropriate therapeutic action. For example, there may be conditions in the patient that prohibit the application of a particular therapy. Because there may be different types of conditions that affect the choice of therapy, it is not possible to be very precise about their nature in the application ontology. Using guideline B.5, it is decided to model therapeutic problems using the general concept `finding`. Although guideline B.5 advises on making a specialization of the general concept, this is not done yet, because it is not clear in what way the specialization could be more specific than `finding` itself.

Therapeutic data. As was the case with diagnostic data, the data that are used to derive the therapeutic problems are coming from the computerized record. Therefore, these data are also modelled using the concept `ars-datum`.

Therapeutic abduction. In the theory therapy, the relation `has-therapy` is defined as a relation between disorders and therapies. Therefore, this relation is used as a first guess for an appropriate abductive relation.

However, in the ARS application the therapies are not related to disorders but to lesion gradings, which are modelled as a sub-type of findings. This is the same situation that occurred with the `manifestation-of` relation for diagnostic abduction. According to guideline B.4, the concept must be copied to ars-application, modified and renamed. The renamed concept is called `ars-has-therapy`.

For modelling the factors that are important for deciding whether a particular therapy is the right therapy for a disorder, therapy provides the relations

has-indicator and has-contra-indicator. These are defined as relations between tuples of has-therapy and findings. However, because the application ontology uses ars-has-therapy instead of has-therapy, has-indicator and has-contra-indicator must also be modified and copied to ars-application. The modified concepts are named ars-has-indicator and ars-has-contra-indicator.

Therapeutic abstraction. There are many different kinds of therapeutic abstraction with varying degrees of complexity. Thus, the same arguments that were used for modelling the diagnostic abstractions in broad terms, are also applicable for therapeutic abstraction. For this reason, the ars-abstracted-from relation is used again to describe the relation between data and therapeutic problems.

Figure 45 shows a large part of the application ontology for the ARS application, using QUOTE's graphical representation. The figure also illustrates which parts of the ontology are library concepts, specializations of library concepts or modifications of library concepts and which concepts are new.

7.4. EXTENDING THE LIBRARY

As described in Section 3, the concepts that were newly defined for the ARS application can be used to extend CUE's ontology library. To do this, they must be scored on the domain-specificity index and the method-specificity index. In cases where the present domain and the used methods are not in the domain and method hierarchies, these must also be added. Table 4 shows how the newly defined concepts were scored on the domain- and method-specificity attributes.

Domain specificity. In Section 7.3 it was mentioned that the current domain—ARS management—is related to four medical sub-domains: haemotology, dermatology, neurology and gastroenterology, but that there are as yet no entries for these sub-domains in the library. In this situation, there are two options. We could add the sub-domains to the domain hierarchy and make ARS management a specialization of each of them. This strategy is appropriate when there are new concepts in the application ontology that are specific to these subdomains. However, inspection of the concepts in ars-application shows that this is not the case. The other option is to make ARS management a direct specialization of the domains from which it uses concepts. As was explained in the previous section, most of the newly defined concepts are specializations or modifications of core library concepts. The other newly defined concepts (ars-abstracted-from and ars-indicator-value) were defined from scratch. For this reason it is decided to make ARS management a direct specialization of ''medicine'', the root of the domain hierarchy.

Now it must be decided whether the newly defined concepts are specific for ARS management or whether they are generic for the medical domain. For the concepts that are specializations or modifications of core library concepts it is evident that they are specific to ARS management. Therefore this domain becomes their domain specificity value. Also ars-indicator-value gets ARS treatment as its domain-specificity value. This concept is only intended for modelling the set of possible values for ars-lesion-indicator. Because the two concepts are closely related, they should have the same values on the domain- and method-specificity attributes.

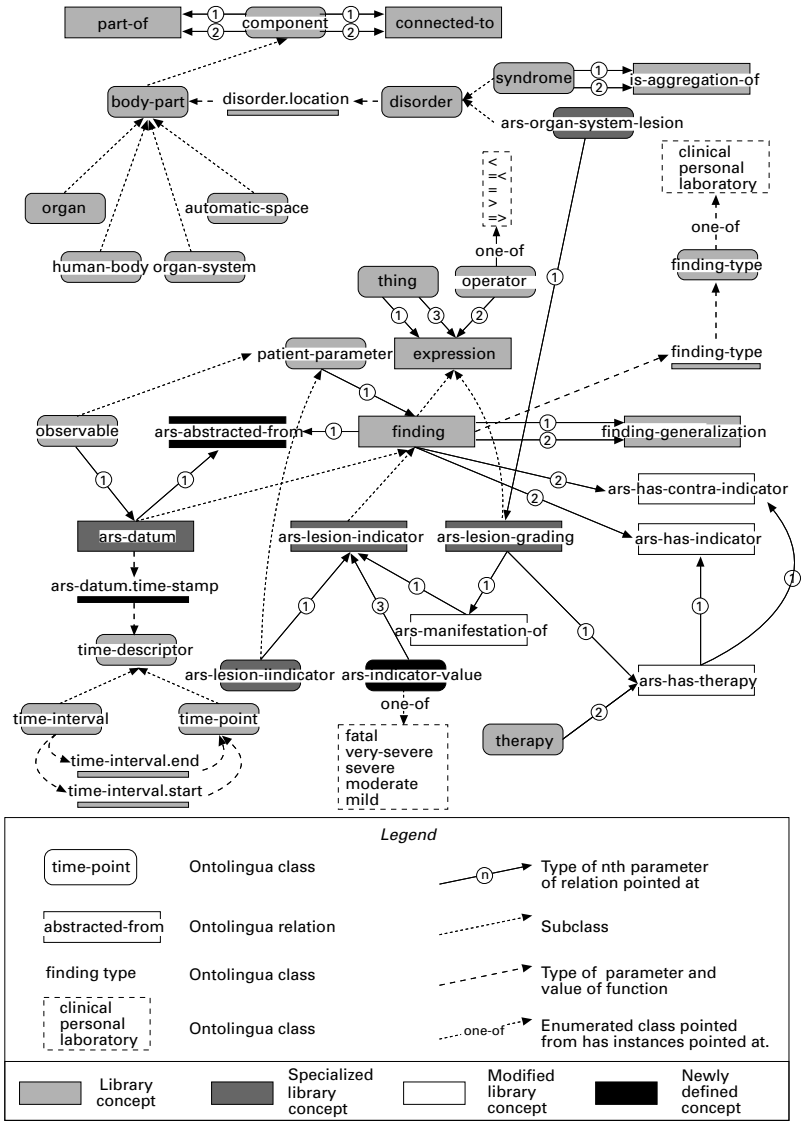

FIGURE 45. The ARS ontology.

The concept `ars-abstracted-from` was used for modelling the situation where the abstractions are made using diverse (mathematical) procedures which are not directly related to the expertise required for problem solving. Because this is a situation which is often found in medicine, it is decided to rename this concept as `abstracted-from` and assign the value ''medicine'' to it for the domain-specificity attribute.

Method specificity. In the ARS application two types of methods were used: ''abduction by direct associations'' and ''abstraction by invoking mathematical procedures''. Both methods are represented in the method hierarchy. To score the

TABLE 4

The domain- and method-specificity values of the concepts that were newly defined in the ARS application. Abduction by D.A. stands for abduction by direct associations, and Abstraction by M.P. stands for abstraction by mathematical procedures

Concept	Domain specificity	Method specificity
ars-datum	ARS Management	Medical method
ars-datum.time-stamp	ARS Management	Medical method
ars-lesion-indication	ARS Management	Medical method
ars-indicator-value	ARS Management	Medical method
ars-has-therapy	ARS Management	Abduction by D.A.
ars-has-indicator	ARS Management	Abduction by D.A.
ars-has-contra-indicator	ARS Management	Abduction by D.A.
ars-manifestation-of	ARS Management	Abduction by D.A.
abstracted-from	Medicine	Abstraction by M.P.
ars-lesion-grading	ARS Management	Medical method
ars-lesion-indicator	ARS Management	Medical method
ars-organ-system-lesion	ARS Management	Medical method

concepts on this attribute, the following guideline is used. When a concept plays a primary role in the reasoning process (e.g. hypothesis, datum or patient finding in diagnosis), it gets the method-specificity value ''medical method'', which is the root of the method hierarchy. If the concept is used for an inference, the method specificity value of the concept is similar to the method specificity value of the concept that it is a specialization or modification of, except when the specialization or the modification was introduced for enabling the use of a more specialized method. In the latter case, the specialized method is used as method-specificity value.

7.5. MAPPING TASK MODEL AND ONTOLOGY

Because application ontology construction was initiated by determining the ontological features of the concepts that play the primary roles in the reasoning process, the mapping between the roles in the STModels and the ontology is straightforward. As illustrated in Figure 46, QUITE has specialized editors for defining the mappings between the task model components and the concepts in the application ontology. The mappings are summarized in Table 5 and Table 6. The tables also indicate how the ontological concepts were included in the application ontology: (i) directly from the library (library), (ii) by specializing a library concept (specialized), (ii) by making a modified copy of a library concept (modified) or (iv) defined from scratch (new).

In Section 7.2, the two generic-task instances were connected by means of a control link. When the ontology mappings have been defined, it is also possible to specify the role-to-role mappings. The purpose of these role-to-role mappings is to specify how the knowledge roles of the different generic-task instances in the task model are related. If there is a role-to-role mapping between two knowledge roles,

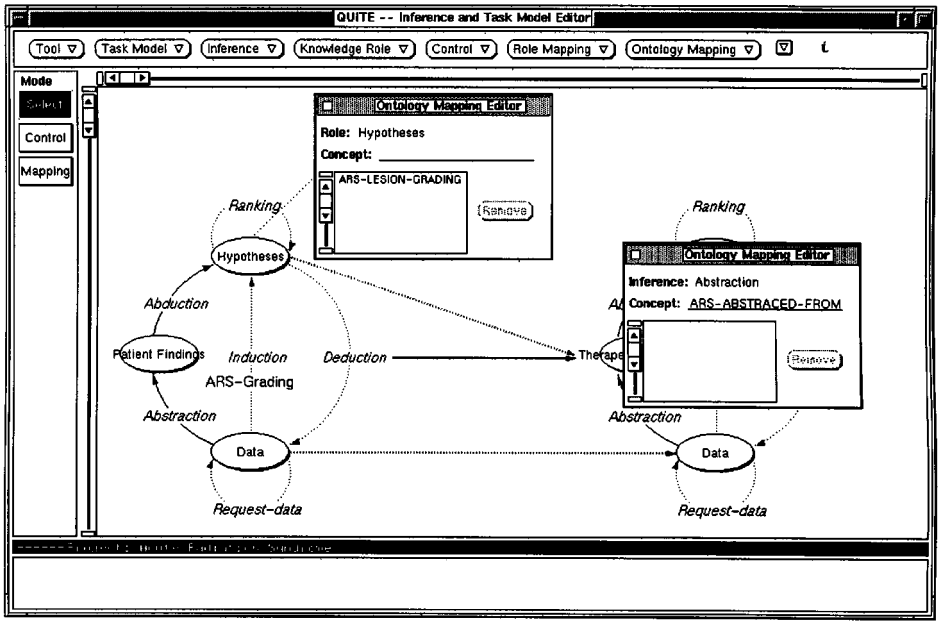

FIGURE 46. Ontology mappings and role-to-role mappings in the ARS task model. The figure shows that the diagnostic hypotheses are mapped onto the ontological type `ars-lesion-indication` and that the therapeutic abstraction inference is mapped onto the relation `ars-abstracted-from`. Further, the figure shows role-to-role mappings between diagnostic hypotheses and therapeutic problems and between diagnostic therapeutic data.

a piece of knowledge that plays the role that the role-to-role mapping points from automatically also plays the role that the mapping points to. As can be seen in Figure 46, there are two role-to-role mappings in the ARS task model: diagnostic data are mapped onto therapeutic data and diagnostic hypotheses are mapped onto therapeutic problems. The first of these mappings is essential for the reasoning process. It explicates that the hypothesized diagnoses are therapeutic problems,

TABLE 5

Ontology mappings between the diagnostics STModel of the task model and the application ontology. The table also shows how the ontological concepts were included in the application ontology

Diagnostic role	Ontological concept	How included
Data	`ars-datum`	Specialized
Patient findings	`ars-lesion-indication`	Modified
Hypotheses	`ars-lesion-grading`	Specialized
Abstraction	`ars-abstracted-from`	New
Abduction	`ars-manifestation-of`	Modified
Ranking	—	—
Deduction	—	—
Induction	—	—

TABLE 6
Ontology mappings between the therapeutic STModel of the task model and the application ontology. The table also shows how the ontological concepts were included in the application ontology

Therapeutic role	Ontological concept	How included
Data	ars-datum	Specialized
Therapeutic problems	finding	Library
Hypotheses	therapy	Library
Abstraction	ars-abstracted-from	New
Abduction	ars-has-therapy	Modified
	ars-has-indicator	Modified
	ars-has-contra-indicator	Modified
Ranking	–	–
Deduction	–	–
Induction	–	–

thereby connecting the two generic task instances. The other role-to-role mapping is for convenience. It explicates that the data that are used for ARS grading may also be used for planning a therapy. For the current application, this has no impact since both the diagnostic data and the therapeutic data are retrieved from the same database. However, such mappings are important when data acquisition is a laborious task.

7.6. ACQUIRING DOMAIN KNOWLEDGE

7.6.1. Generating the elicitation agenda

For the acquisition of the actual domain knowledge QUAKE is invoked. When the tool is started, its first task is to generate an initial elicitation agenda. For generating this agenda it must be determined which of the concepts in the application ontology need to be instantiated in the knowledge base, and which are only used for adding higher-level structure to the application ontology.

This decision can be made manually or automatically in QUAKE. When performed manually, the tool presents all the classes, relations and functions and invites the user to specify which of these concepts will have instances or tuples in the knowledge base. In automatic mode, QUAKE uses the mappings between the task model and the application ontology to decide which classes and relations need to be instantiated. The underlying idea is that only the concepts mentioned in those mappings are used in the actual reasoning process. Therefore, these are the only ones that can affect the behaviour of the KBS. For the ARS application, the initial agenda is generated automatically.

7.6.2. Defining the knowledge elicitation strategy

When it has been decided which knowledge must be elicited, the next step is to decide in which order the knowledge must be elicited. In QUOTE this is specified by means of a knowledge elicitation strategy.

In Section 5, two general principles were formulated for specifying knowledge

elicitation strategies. Firstly, it was observed that there are usually *basic objects* in an application domain. For medical domains these are typically disorders, and in engineering domains components. Secondly, it was observed that the elicitation strategy is usually based on some kind of graph traversal: already elicited knowledge is used to prompt for related knowledge. From these principles and the application ontology in the ARS domain the following elicitation strategy was derived.

(1) Elicit the diagnostic hypotheses (`ars-lesion-gradings`).
(2) For each of the elicited hypotheses, use the main abductive relation to elicit the abstract findings that would trigger that hypothesis. In this case, this is the relation `ars-manifestation-of`. Note that the abductive knowledge and the patient findings (`lesion-indications`) are elicited in one step.
(3) For each of the abstract findings, elicit the knowledge used to make the abstractions (the `ars-abstracted-from` tuples and the instance of `ars-datum`).
(4) Elicit for each of the diagnostic hypotheses the associated therapies, using the basic therapeutic abduction relation (`ars-has-therapy`).
(5) Elicit the findings that are indicators and contra-indicators for using a particular therapy for a lesion grading.
(6) Elicit for each of the findings that are indicators or contra-indicators for the therapies the data that they are abstracted from.

Note that in this strategy a number of ordering decisions have been taken which are not derived from the principles mentioned above. For example, the principles do not suggest that the knowledge used for diagnosis should be elicited before the knowledge used in therapy planning. In other words, it is not possible to derive a unique best strategy from the principles. There are a number of sensible strategies possible. From these, one has been chosen arbitrarily. Figure 47 shows how this strategy was formulated in QUAKE's knowledge-elicitation-strategy language.

7.6.3. Eliciting the application knowledge

Given the application ontology and the knowledge elicitation strategy, elicitation is a straightforward activity. QUAKE prompts the domain expert with a series of questions of the type ''Enter a possible grading of the haemopoietic lesion''. Figure 48 shows a transcript of the part of the scenario where the system elicits the indicators and contra-indicators of therapies.

7.7. BUILDING THE DESIGN MODEL

When the knowledge model has been completed, it must be transformed into an executable system. In our approach, the design process consists of three steps: (i) implementing the problem solving method, (ii) selecting problem solvers and (iii) translating the knowledge. This section will concentrate mainly on the first two of these steps. When the problem solvers are selected and the mappings between the application ontology and the representational meta models have been specified, the translation step can be done automatically.

```
(define-ka-strategy main ( )
  ; ; Elicit the organ system lesions
  (elicit-all ?lesion (ars-organ-system-lesion ?lesion)
    (elicit-all ?grading (ars-lesion-grading ?lesion=?grading)))
  ; ; For each of the elicited hypotheses, use the abductive relation
  ; ; to elicit the findings that would trigger that hypothesis.
  (for-each $1g (ars-lesion-grading $1g)
    (elicit-all $1i (manifestation-of (ars-lesion-indication $1i)
                                      (ars-lesion-grading $1g))))
  ; ; For each of the abstract findings, elicit the knowledge that is
  ; ; used to make the abstractions
  (for-each $1i (ars-lesion-indication $1i)
  (elicit-all $datum (ars-abstracted-from (ars-lesion-indication $1i)
                     (ars-datum $datum))))
  ; ; Elicit for each of the diagnostic hypotheses the associated
  ; ; therapies, using the basic therapeutic abduction relation
  (for-each $1g (ars-lesion-grading $1g)
    (elicit-all ?t (ars-has-therapy (ars-lesion-grading $1g)
                                    (therapy ?t))))
  ; ; Elicit the findings that are indicators and contra-indicators for
  ; ; using a particular therapy for a lesion grading.
  (let (@findings)
    (for-each $aht (ars-has-therapy $aht)
      (elicit-all $finding (ars-has-indicator $aht $finding)
        (push $finding @findings))
      (elicit-all $finding (ars-has-contra-indicator $aht $finding)
        (push $finding @findings)))
  ; ; Finally, elicit the data from which the findings are abstracted.
  (for-each $finding (member-of $finding @findings)
    (elicit-all $datum (ars-abstracted-from (finding $finding)
                                            (ars-datum $datum))))))
```

FIGURE 47. The knowledge elicitation strategy used for the ARS application. The language used for defining strategies distinguishes three types of variables: (i) instance variables (`?varnames`), which unify with class instances, (ii) tuple variables (`$varname`), which unify with tuples of the indicated types and (iii) set variables (`@varname`) which can be used for the temporary storage of elicited instances and tuples. The basic constructs in the language are `elicit-all` and `for-each`. Both have as their first parameter a variable, and as their second parameter an expression that constrains the instances and tuples with which the variable can unify. The (optional) remaining arguments must be operations that are to be performed on each of the instances or tuples that are unified with the variable. The difference between `elicit-all` and `for-each` is that the former obtains the objects that are unified with the variable by asking the QUAKE user, whereas the latter searches for objects that unify with the variable in QUAKE's knowledge repository. Further, the ARS strategy uses the constructs `push`, which includes-an object in a set, and let, for (lexical) variable scoping. Besides these constructs, the language also provides simple constructs for conditional branching, supports user-defined procedures and allows recursion.

7.7.1. Implementing the problem-solving method

The problem-solving method has been implemented by means of meta rules which specify the high-level control of the reasoning process. This involves both sequencing of the inferences within an instantiated task model and switching between these models.

At the highest level, it must be specified that diagnosis is performed before therapy planning. This is the most common situation. Only in time-critical situations it might occur that the two processes are interleaved. However, it is not intended that the ARS system be embedded in a real-time environment. Further, the amount

At a certain moment in the scenario, the system is eliciting therapies for the various lesion-gradings:
`(ars-has-therapy (HPS=4) ⟨therapy⟩)`
`Enter therapy:` autologous-BMT
The user enters that autologous-BMT is a possible therapy for grading 4 of HPS (*haemopoetic syndrome*). *The system continues by asking for possible therapies for every lesion-grading. It then starts asking for the indicators and contra-indicators of these therapies.*
`(ars-has-contra-indicator (ars-has-therapy (HPS=4) autologous-BMT) ⟨finding⟩)`
`Enter Finding:` stem-cells-preserved = no
`(ars-has-contra-indicator (ars-has-therapy (HPS=4) autologous-BMT) ⟨finding⟩)`
`Enter Finding:`
identical-twin-available = no

FIGURE 48. A transcript of a part of the knowledge elicitation session for the ARS application.

of knowledge in the knowledge model is limited and well structured, so it is not expected that the system will have to traverse huge search spaces. The following meta rules specify when the two generic task instances may be invoked. It is assumed that when control is passed to the task invocation module, the presence of hypotheses in the hypotheses space indicates whether the task has been completed.

IF Hypotheses-space of ARS-Grading = empty
THEN invoke-task-instance ARS-grading

IF Hypotheses-space of ARS-Grading = NOT empty **AND**
Hypotheses-space of Select-ARS-Treatment = empty
THEN invoke-task-instance Select-ARS-Treatment

For each of the generic task instances, the local control regimes must also be specified. The suitability of a particular control regime depends on the goal that is associated with the task. For ARS-grading this is ''grading the disease''. Computationally, this is an easy kind of diagnosis, because it allows us to make the single fault assumption: according to the application ontology a lesion can only have one true grading. Because of the single fault assumption and because there is a finite number of possible gradings, the search space is finite. Furthermore, in this particular case the search space is very small. The attractive computational characteristic allow for a straightforward control regime. First the data are obtained, then all the possible abstractions are made, and then the hypotheses are generated. The following meta rules implement this strategy for ARS-grading.

IF Observable-Data-Space = empty
THEN Invoke-inference Request-Data

IF Observable-Data-Space = NOT empty **AND**
Patient-Findings Space = Empty
THEN Invoke-inference Abstraction

IF Hypotheses-space = empty **AND**
Patient-Findings Space = NOT empty
THEN Invoke-inference Abduction

7.7.2. Selecting problem solvers

As argued in Section 6, the problem solvers that implement the inferences should be both epistemologically and computationally adequate. Considering the limited amount of domain knowledge, it is not likely that computational complexity will be an important issue in this case. Therefore, the discussion will focus on epistemological adequacy. A problem solver is epistemologically adequate if the distinctions between different kinds of knowledge in the domain are preserved in the data structures used by the problem solver. In Section 6 this idea was operationalized by means of mappings between the application ontology and representational meta models of the problem solvers. When the mappings are simple, the problem solver is epistemologically adequate.

Diagnostic abduction. Diagnostic abduction is performed by traversing tuples of the relation `ars-manifestation-of`. As can be seen in Figure 45, the abductive inferences are straightforward: the findings are directly associated with the hypotheses that they trigger and no uncertainty is involved. Such associational reasoning can easily be performed by a forward-chaining production-rule interpreter. Therefore, the representational meta model of a simple rule interpreter is selected from CUE's problem solver library. Figure 49 shows this model using QUOTE's graphical representation.

The representational meta model specifies that a rule consists of three sets of attribute expressions: conditions, counter-conditions and actions. Attribute expressions are the symbol-level equivalent of findings in the application ontology. They consist of an attribute, an operator and an attribute-value. The representational meta model makes an explicit distinction between conditions and counter-conditions. A rule may fire when all of its conditions and none of its counter-conditions are true. There are two reasons for making this distinction. Firstly, without this distinction, modelling of attribute expressions would be more complicated, since there would be a need for ''negative'' attribute expressions. Secondly, as will see later, conditions and counter-conditions are often mapped onto different ontological concepts. The semantics of the representational meta model are that a rule may fire when all of its conditions are satisfied and when none of its counter-conditions are satisfied.

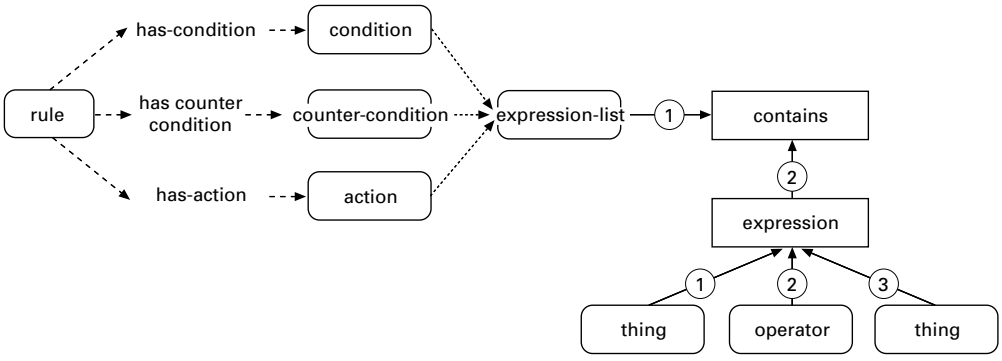

FIGURE 49. The representational meta-model of a production-rule interpreter, represented using QUOTE's graphical language.

```
(ars-manifestation-of (ars-lesion-indication ?1i=?v)
                      (ars-lesion-grading ?1g=?g))
  :→
(and (rule ?rule)
     (has-condition ?rule ?condition)
     (contains ?condition (attribute-expression ?1i patient ?v))
     (has-action ?rule ?action)
     (contains ?action (attribute-expression ?1g patient ?g)))
```

FIGURE 50. A simple mapping from the `ars-manifestation-of` relation onto the representational meta-model of the production rule system. The mappings are specified by means of mapping rules. On both sides of a mapping rule are expressions in CUE's logical notation. The expressions may contain variables. When the mappings rules are applied, every expression that can be unified with the left-hand side of the mapping rule can be rewritten as the right-hand side.

There are different ways to map the relevant parts of the application ontology onto the representational meta model. The main ontological concept for the mapping is the `ars-manifestation-of` relation. A straightforward way of mapping this relation onto the representational meta model is to generate a rule for every tuple of the relation. The mappings could then be specified as shown in Figure 50.

One complication that must be handled is that the second argument of `finding` (of which `ars-lesion-indication` and `ars-lesion-grading` are sub-types) may be either $<$, $=<$, $=$, $>$ or $>=$, whereas in the attribute expressions in the representational meta model it can only be specified that an object has a particular value for an attribute. Thus, only the $=$ operator may be used. Further, there is no explicit representation of the object of the attributes (the patient) in the application ontology. In the mappings in Figure 50 this is handled by enforcing that only findings with the $=$ operator may be mapped and by using the constant `patient` as a dummy object in the attribute expressions in production rules. Inspection of the application knowledge shows that the restriction to the $=$ operator does not cause problems.

A problem with the mapping in Figure 50 is that it generates a large number of hypotheses. As soon as one of the manifestations of a particular grading has been established, the hypothesis will be generated. In the ARS domain a more conservative abductive strategy is preferred: a lesion grading should only be hypothesized if all of its manifestations are present. This can be realized by specifying the mapping rules in such a way that all the manifestations are grouped in the condition part of a single rule.

Note that the representational meta model does not allow disjunctive conditions. Although allowing disjunction does not affect the representational power of the formalism—it is equivalent to the introduction of extra rules—it may affect the mapping relations. For example, if disjunctions are allowed, it is always possible to generate one and only one rule for every possible action. If disjunction is not allowed, this is only possible in cases where every condition is a necessary condition, and if there are no subsets of the set of conditions that are sufficient for the action. In terms of the application ontology, this means that a particular lesion grading can only be the true grading if all of its associated manifestations are present. As described in Section 7.3, this requirement is met in the ARS domain. (It was for this reason that the attributes `frequency` and `evoking-strength` were left out of the

```
(ars-lesion-grading ?1g=?g)
  :→
(and (rule ?rule)
     (has-action ?rule ?action)
     (contains ?action (attribute-expression ?1g patient ?g)))

(ars-manifestation-of (ars-lesion-indication ?1i=?v)
                      (ars-lesion-grading ?1g=?g))
  :→
(⇒ (and (rule ?rule)
         (has-action ?rule ?action)
         (contains ?action (attribute-expression ?1g patient ?g))
         (has-condition ?rule ?condition))
    (contains ?condition (attribute-expression ?1i patient ?v)))
```

FIGURE 51. The mapping rules that are used for diagnostic abduction in the ARS application.

application ontology.) The mapping is realized by means of the rules shown in Figure 51.

The first mapping rule in Figure 51 states that for every lesion grading in the knowledge model there is a rule in the design model, where the action part of the rule asserts a particular lesion-grading. The second mapping expresses that for every tuple of the `ars-manifestation-of` relation which has the particular lesion-grading as its second argument, the rule that is based on that grading has a condition that corresponds to the first argument of the tuple.

Diagnostic abstraction. In the application ontology, the abstractions are modelled by means of `ars-abstracted-from` relations between findings and lesion indications. In the design model, we must specify the procedures that compute the abstractions and build a wrapper around these procedures to ensure that they can be invoked in the same way as "real" problem solvers. As mentioned, the complexity of the abstractions in the ARS domain ranges from simple table lookups to complex computational procedures. We will concentrate here on one abstraction of each type.

An example of a simple qualitative abstraction is the derivation of the lesion-indicator `vomiting-severity` from the data `vomiting-time` and `accident-time`. The value of this indicator must be one of five time periods, which represent the time period in hours between the exposure to radiation and the moment of vomiting. The Lisp function shown in Figure 52 is used to compute the vomiting severity.

```
(defun compute-vomiting-severity (accident-time vomiting-time)
  (let ((period (- vomiting-time accident-time)))
    (cond ((< period 9.2)
           '(0 0.2))
          ((< period 0.5)
           '(0.2 0.5))
          ((< period 2.0)
           '(0.5 2.0))
          ((< period 6.0)
           '(2.0 6.0))
          (t '(6.0: infinity)))))
```

FIGURE 52. The Lisp function that computes the vomiting severity abstraction.

A complex kind of abstraction is the derivation of the severity of the granulocyte decrease. This lesion indicator, which is an indicator of the severity of the lesion to the haemopoietic system, is abstracted from a measurement of the granulocytes concentration and the measurement time relative to the date of the radiation exposure. The severity of the decrease is expressed by a number between 1, meaning not severe, and 5, meaning very severe. The value of the granulocyte decrease is computed by comparing the measured granulocyte concentration with time-dependent threshold values. The computation of the threshold values is based on a quantitative model of the development of the granulocytes concentration, which can be expressed by means of the function g:

$$g(t) = \frac{a}{-a+b} \cdot e^{-at} \cdot x + \frac{a}{a-b} e^{-bt} \cdot x + a \cdot e^{-bt} \cdot y$$

The model is based on four parameters: x, the number of maturing granulocytes in the home marrow; y, the number of granulocytes in the blood; a, the maturation time of granulocytes and b, the loss rate of granulocytes. The severity of the haemopoietic lesions can be modelled in terms of values for the four parameters in the model. To generate the functions that discriminate between the five levels of severity of the haemopoietic lesion, the parameter values shown in Table 7 are used. (These values are based on empirical results.)

The Lisp function that computes the granulocyte-decrease abstraction is shown in Figure 53.

To embed the Lisp function in the KBS they must be wrapped in an object that can be invoked as a problem solver. In the current application this is realized by wrapping the abstraction functions into production rules. The conditions of these production rules are used to bind the parameters of the Lisp functions to particular values, and the actions are used to write the function result to the appropriate space of the blackboard.

Diagnostic data entry. In the ARS application, the diagnostic data are derived from computerized medical records. In Section 7.3 it was explained that the diagnostic data are modelled by means of `ars-datum`, a specialization of `finding`. However, the data are not represented in this format in the medical record.

In order to enable the ARS application to make use of the data in the database, a transformation must be defined between the `ars-datum` concept and the database

TABLE 7
Parameter values for the threshold functions to discriminate between different values of the granulocyte-decrease abstraction

Degree	x	y	a	b
1–2	101	5	1	2.4
2–3	120	5	0.475	2.4
3–4	105	5	0.166	2.4
4–5	100	5	0.062	1.185

```
((defun compute-granulocyte-decrease (concentration time)
  (flet ((threshold-function (x y a b)
           (+ (* (/ a (- b a))
                 (exp (* -1 a time))
                 x)
              (* (/ a (-a b))
                 (exp (* -1 b time))
                 x)
              (* a
                 (exp (* -1 b time))
                 y))))
(cond ((< concentration (threshold-function 101 5 1 2.4))
      5)
      ((< concentration (threshold-function 120 5 0.475 2.4))
      4)
      ((< concentration (threshold-function 105 5 0.166 2.4))
      3)
      ((< concentration (threshold-function 100 5 0.062 2.4))
      (t 1))))
```

FIGURE 53. The Lisp function that computes the granulocyte decrease abstraction.

schema. As with problem solvers, this is done in two steps: first, make a mapping between the ontology and a representational meta model of the database, and then use a translation program to translate between the representational meta model and the actual database representation.

For the present purpose, a simple database representational meta model is used: a database consists of sections, and each of these sections consists of a number of records. In turn, records consist of a number of fields. Different sections of the database may have different kinds of records, but, within a section, all records must have the same type of fields. Figure 54 shows the mappings between the representational meta model and the application ontology for the erythema part of the database (see Table 3).

The mapping between the database records and the application ontology has a different nature from the mappings between the ontology and the representational meta models of the problem solvers described earlier. In contrast with the earlier mappings, the mapping shown in Figure 54 is specific for one particular kind of `ars-datum`, namely data about erythema. For each of the different kinds of data used in the application another mapping must be defined. This is necessary because the ARS database uses a variety of record structures.

```
(has-record erythema-section
            ?erythema-location ?yes no ?unknown ?begin ?end ?maximum ?degree)
  :→
(ars-datum ?erythema-location=no)

(has-record erythema-section
           ?erythema-location yes ?no ?unknown ?begin ?end ?maximum ?degree)
  :→
(and (ars-datum ?erythema-location=?degree)
     (ars-datum.time-stamp (ars-datum ?erythema-location=?degree)
                           (time-interval ?begin ?end)))
```

FIGURE 54. The mapping rules used for diagnostic data entry.

```
(ars-has-therapy (lesion-grading ?l1=?v) ?t)
  :→
(and (rule ?rule)
     (has-action ?rule ?action)
     (contains ?action (attribute-expression therapy patient ?t))
     (has-condition ?rule ?condition)
     (contains ?condition (attribute-expression ?l1 patient ?v)))

(ars-has-indicator (ars-has-therapy (lesion-grading ?l1=?g) ?t)
                   (finding ?ob=?v))
  :→
(⇒ (and (rule(and ?rule)
         (has-action ?rule ?action)
         (contains ?action (attribute-expression therapy patient ?t))
         (has-condition ?rule ?condition)
         (contains ?condition (attribute-expression ?l1 patient ?g))
    (contains ?condition (attribute-expression ?ob patient ?v)))

(ars-has-contra-indicator (ars-has-therapy (lesion-grading ?l1=?g) ?t)
                          (finding ?ob=?v))
  :→
(⇒ (and (rule ?rule)
         (has-action ?rule ?action)
         (contains ?action (attribute-expression therapy patient ?t))
         (has-condition ?rule ?condition)
         (contains ?condition (attribute-expression ?l1 patient ?g))
         (has-counter-condition ?rule ?counter-condition))
    (contains ?counter-condition (attribute-expression ?ob patient ?v)))
```

FIGURE 55. The mapping rules used for therapeutic abduction in the ARS application.

Therapeutic abduction. For the therapeutic abduction inference, three relations are used: `ars-has-therapy`, `ars-has-indicator` and `ars-has-contra-indicator`. `ars-has-therapy` directly connects the therapeutic problems to the therapies. This suggests that, as for diagnostic abduction, the production rule interpreter might be suitable. However, for therapy planning the situation is more complicated because we must also deal with the indicators and the contra-indicators.

The indicators are conditions that must hold for the therapy to be appropriate. It is clear that these can be realized computationally as additional conditions in the production rules. The contra-indicators can be realized as counter-conditions in the production rules. Figure 55 shows the mapping rules that implement this operationalization.

Therapeutic abstraction and data entry. We can be brief about the problem solvers that are selected for therapeutic abstraction and therapeutic data entry. Both inferences make the same ontological commitments as the corresponding steps in the diagnostic sub-task, and are therefore implemented in a similar way.

Table 8 summarizes which problem solvers were selected to implement the inferences in the ARS task model.

7.7.3. Translating the knowledge

Once the mappings between the ontology and the representational meta models have been specified, translating the knowledge to the particular formalisms is largely an automatic process. As was mentioned in Section 6, the representational meta

TABLE 8
Problem solvers used in the ARS application

Inference	Problem solver
Diagnostic request data	ARS Database
Diagnostic abstraction	Lisp functions
Diagnostic abduction	production rules
Therapeutic request data	ARS Database
Therapeutic abstraction	Lisp functions
Therapeutic abduction	production rules

models are associated with procedures that translate back and forth between the representational meta models and the internal representations of the problem solvers. The mappings between the application ontology and the representational meta models can therefore be considered as a specification of how the knowledge in the knowledge model should be represented in the design model. For example, the knowledge pieces which were elicited in Figure 48 are translated according to the mapping rules shown in Figure 55 into the following attribute expression in terms of the representational meta model:

```
(rule rule34)
(has-action rule34 action34)
(contains action34 (attribute-expression therapy patient
 autologous-BMT))
(has-counter-condition rule34 counter-condition34)
(contains counter-condition34
    (attribute-expression identical-twin-available patient no))
(contains counter-condition34)
    (attribute-expression stem-cells-preserved patient no))
```

These attribute expressions are then translated automatically into production rules in the syntax of the particular production rule interpreter:

IF NOT identical-twin-available patient no **AND**
NOT stem-cells-preserved patient no
THEN therapy patient autologous-BMT

7.8. QUAARS IN ACTION

To illustrate how the final system solves problems in the ARS domain, we now present an execution trace where the system diagnoses the haemopoietic syndrome. In the trace, the system retrieves eight pieces of data from the database and uses these to abstract the severity of the vomiting reaction, the severity of the diarrhoea reaction and the severity of the granulocyte decrease. Based on these findings, the system derives that the haemopoietic syndrome has grading 4.

```
>(quaars)
;;;-----------------------------------
;;;QSHELL (Version 0.1)
;;;-----------------------------------
------Invoking Task ARS-GRADING
```

```
------ Invoking Inference REQUEST-DATA
------ Retrieving data from database
Retrieved: (ars-datum radiation-exposure=yes)
Retrieved: (ars-datum.time-stamp
           (ars-datum radiation-exposure=yes)
           (time-point ⟨1965/04/28/23.11⟩))
Retrieved: (ars-datum vomiting=yes)
Retrieved: (ars-datum.time-stamp
           (ars-datum vomiting=yes)
           (time-point ⟨1965/04/28/23.52⟩))
Retrieved: (ars-datum diarrhea=yes)
Retrieved: (ars-datum.time-stamp
           (ars-datum diarrhea=yes)
           (time-point ⟨1965/04/28/23.40⟩))
Retrieved: (ars-datum granulocyte-count=3.5)
Retrieved: (ars-datum.time-stamp
           (ars-datum granulocyte-count=3.5)
           (time-point ⟨1965/05/02/12.30⟩))
------ Invoking Inference ABSTRACTION
------ Mapping inputs from DATA space to Representational
Meta Model of ABSTRACTION-WRAPPER
Mapped: (ars-datum radiation-exposure=yes) :→
        (attribute-expression radiation-exposure patient yes)
Mapped: (ars-datum.time-stamp
        (ars-datum radiation-exposure=yes)
        (time-point ⟨1965/04/28/23.11⟩)) :→
        (attribute-expression time radiation-exposure
          ⟨1965/04/28/23.11⟩)
Mapped: (ars-datum vomiting=yes) :→
        (attribute-expression vomiting patient yes)
Mapped: (ars-datum.time-stamp
        (ars-datum vomiting=yes)
        (time-point ⟨1965/04/28/23.52⟩)) :→
        (attribute-expression time vomiting ⟨1965/04/28/23.52⟩)
Mapped: (ars-datum diarrhea=yes) :→
        (attribute-expression diarrhea patient yes)
Mapped: (ars-datum.time-stamp
        (ars-datum diarrhea=yes)
        (time-point ⟨1965/04/28/23.40⟩)) :→
        (attribute-expression time diarrhea ⟨1965/04/28/23.40⟩)
Mapped: (ars-datum granulocyte-count=3.5) :→
        (attribute-expression granulocyte-count patient 3.5)
Mapped: (ars-datum.time-stamp
        (ars-datum granulocyte-count=3.5)
        (time-point ⟨1965/05/02/12.30⟩)) →
        (attribute-expression time granulocyte-count
          ⟨1965/05/02/12.30⟩)
```

```
------ Translating inputs to representation of ABSTRACTION-WRAPPER
Finished
------ Invoking Problem solver ABSTRACTION-WRAPPER
Finished
------ Translating outputs of ABSTRACTION-WRAPPER to Representational
Meta Model
Finished
------ Mapping outputs from Representational Meta Model of
ABSTRACTION-WRAPPER to PATIENT-FINDINGS
Mapped: (attribute-expression vomiting patient severe) :→
        (ars-lesion-indication vomiting=severe)
Mapped: (attribute-expression diarrhea patient severe) :→
        (ars-lesion-indication diarrhea=severe)
Mapped: (attribute-expression granulocyte-decrease patient severe) :→
        (ars-lesion-indication granulocyte-decrease=severe)
------ Invoking Inference ABDUCTION
------ Mapping inputs from PATIENT-FINDINGS space to Representational
Meta Model of Q-CHAIN
Mapped: (ars-lesion-indication vomiting=severe) :→
        (attribute-expression vomiting patient severe)
Mapped: (ars-lesion-indication diarrhea=severe) :→
        (attribute-expression diarrhea patient severe)
Mapped: (ars-lesion-indication granulocyte-decrease=severe) :→
        (attribute-expression granulocyte-decrease patient severe)
------ Translating inputs to internal representation of Q-CHAIN
Finished
------ Invoking Problem Solver Q-CHAIN
Finished
------ Translating outputs of Q-CHAIN to Representational Meta Model
Finished
------ Mapping outputs from Representational Meta Model of Q-CHAIN to
HYPOTHESES space
Mapped: (attribute-expression HPS patient 4) :→
        (ars-lesion-grading HPS=4)
```

8. Conclusions

An ontology is an explicit, knowledge level specification of a conceptualization. This paper has described a number of ways in which ontologies can be used in the knowledge engineering process.

To maximize the leverage of explicit ontologies, an application-specific ontology should be constructed early in the knowledge engineering process. We have called such an ontology an application ontology. However, in Section 3 it was argued that the contents of an application ontology may depend heavily on the task at hand. Therefore, application ontologies can more easily be constructed when a task model

is at hand. On the basis of these observations we have proposed the following organization of the knowledge engineering process.

(1) Construct a task model for the application.
(2) Select and/or construct appropriate ontologies, and if necessary refine these.
(3) Map the application ontology onto the knowledge roles in the task model.
(4) Instantiate the application ontology with domain knowledge.
(5) Select or construct meta rules that implement the task model.
(6) Select problem solvers that implement the inferences in the task model.
(7) Translate the domain knowledge into the representations of the selected problem solvers.

Ontologies can be constructed by selecting and configuring ontological theories from a library and by defining ontologies ''from scratch''. To guide navigation in a library of reusable ontological theories, the theories should be indexed according to domain specificity and method specificity. The underlying idea is that some concepts are more reusable than others, and that the reusability of concepts depends on (i) how specific these are for particular domains and (ii) how specific these are for particular problem-solving methods.

An ontology library should be organized in a core part, containing definitions of the basic concepts in the field, and a peripheral part, containing definitions that are only needed for specific methods and specific sub-domains. By indicating which sub-domains and which methods are to be used in an application, the library indexes can be used to find definitions that are likely to be useful for the application. The core part is thus specific for a field (e.g. medicine) but generic across all specializations and tasks within that field.

Often, the library will not contain entries for the sub-domain and for the methods used by an application. For this situation, a number of guidelines have been developed for using the library as a source of inspiration. These guidelines were based on the considerations that (i) only concepts that are referred to by the task-model (and the concepts that they depend on) are needed for an application ontology, (ii) similar concepts will often have similar names, and (iii) it is easier to define new concepts by specifying or modifying existing concepts than to define concepts from scratch.

When there is no library available, application ontologies must be developed by the knowledge engineer. For defining an ontological concept, the following guidelines can be used: (i) a concept should be sufficiently general to cover all the elements of knowledge that the concept is intended for, (ii) a concept should be sufficiently specific to cover only those elements of knowledge that the concept is intended for, and (iii) a concept should have a name that is meaningful in the application domain.

We have described two ways in which explicit ontologies can be used during knowledge engineering: for knowledge elicitation and for computational design. Because ontologies specify which constraints domain knowledge should satisfy, they can be used to direct the knowledge elicitation process. Further they can be used to drive an automated knowledge elicitation tool. In Section 5 this was illustrated with QUAKE, a tool in the CUE workbench, which is able to inspect application ontologies written in Ontolingua. QUAKE uses a knowledge-acquisition oriented interpretation

of the vocabularly defined in the Frame ontology—a representational ontology which defines the vocabularly for specifying Ontolingua ontologies—to support model instantiation by consistency checking, completeness checking, using domain-specific terminology, intuitive visualization and dialogue structuring.

During computational design application ontologies can be used to determine the suitability of problem solvers for the particular application. To decide whether a particular problem solver is suitable, its representational capacity must be compared with the epistemological distinctions in the domain knowledge. The application ontology is an explicit representation of these epistemological distinctions.

An important final question raised in this paper concerns the reusability of ontologies. Handling the *interaction problem* was identified as a key to reusability. The interaction problem states that the way in which knowledge is represented is determined by knowledge use. The interaction problem can be managed by means of the *explicit interaction principle* as follows.

> Different elements of an ontology are affected in different ways by the nature of the method that is used by an intelligent agent. By making the nature of the interaction between the method and the elements of the ontology *explicit,* it can be determined under which conditions ontological elements can be reused.

In Section 3 the explicit interaction principle was used to organize an ontology library. The method-specificity index was used here to make explicit with which method or group of methods an ontological element could be (re-)used.

Most ontology-related research in AI is done for the purpose of knowledge sharing. Knowledge sharing means that a knowledge base can act as a knowledge provider (a server) for other knowledge bases (clients). An influential research project in this context is the knowledge sharing effort (KSE) described in Neches *et al.* (1991). In an analysis of the obstacles that prohibit knowledge sharing between existing knowledge bases, one of the problems is that servers and clients can make different ontological commitments. The solution proposed in the KSE project is to develop a library of standardized ontologies, and to enforce KBS developers to adhere to these ontological standards. To help KBS developers to comply with the standards, the Ontolingua language and support software were developed. The Ontolingua language is the language which is used for encoding the library of ontologies. This language was also used for ontological modelling in the work described here. The Ontolingua program consists of a collection of translation routines that translate ontologies formulated in the Ontolingua language into the representation formalisms of a number of different problem solvers.

The view on knowledge engineering that underlies the Ontolingua approach is that knowledge engineers first decide which ontological commitments must be made, then use the ontology library to make these commitments explicit, and then use their favourite tool for developing the knowledge base. Roughly speaking, this approach is similar to the approach to knowledge engineering advocated in this paper, although in the Ontolingua approach many subtleties of the use of reusable ontologies for knowledge engineering remain unaddressed. For example, in theory the builders of the KSE ontology library do not take into account that a problem-solving method may require particular ontological commitments. That is, the approach does not provide guidelines for dealing with the interaction problem.

In practice, however, the importance of the interaction problem is certainly recognized, as can be seen in the Ontolingua theories developed as part of the VT experiment (Schreiber & Birmingham, 1996). The ontology for this application is split into two theories: one which models concepts that are specific to elevators (the domain), and one that is specific to engineering design (the task). This is exactly what the principles put forward in Section 3 prescribe.

A potential problem with the Ontolingua approach is that it requires that the knowledge bases are based on the same ontological commitments. Therefore, knowledge can only be shared between knowledge bases developed with that very purpose in mind. Unfortunately, the world is filled with knowledge bases and databases which are not developed according to this philosophy. The information in these servers cannot be shared. The approach to this problem presented in this paper was to wrap an external database into an ontological theory which conformed to the commitments in the application ontology. The wrapper was then connected to the database by means of access functions.

A more generic solution to this problem is to use more flexible ways of specifying which ontological commitments are made in a knowledge base. In both the KSE library and our library, committing to the ontological distinctions defined in a theory means including that theory. That is, making the concepts in the included theory directly accessible to the includer. A more flexible way of connecting ontologies is to allow *ontology mappings*. The idea here is that knowledge bases have a base ontology and a number of ontologies that are developed for specific uses of the knowledge base. These use-specific ontologies are then connected to the knowledge base by means of mappings between the base ontology and the use-specific ontologies. The mappings, which specify different viewpoints on the contents of a knowledge base, can be used to reformulate the ontological commitments in the knowledge base in such a way that it is possible to share knowledge with another knowledge base. This approach is currently being investigated in the European KACTUS project (Schreiber, Wielinga & Jansweijer, 1995).

Parts of this article are based on earlier publications with other co-authors. In particular, Section 2 is partially based on an article which was co-authored by Giordano Lanzola and Mario Stefanelli and published in Knowledge Acquisition. Section 3 is based on an article published in Artificial Intelligence in Medicine and was co-authored by Ameen Abu-Hanna, who did the analysis of CASNET, Sabina Falasconi, who developed the core part of the ontology library and Mario Stefanelli. Section 6 is based on a paper presented at the European Conference on Artificial Intelligence (ECAI) '94 and is co-authored by Wilfried Post. Hauke Kindler and Dirk Densow provided the domain knowledge for the acute radiation syndrome application.

We are grateful to Lynda Hardman, Manfred Aben, Peter Terpstra, Anjo Anjewierden, Jan Wielemaker and Frank van Harmelen for their comments on earlier versions of (parts of) this article, and we thank Nicolaas Mars, Nigel Shadbolt, Joost Breuker, Robert de Hoog and Pieter de Vries Robbé for their comments on the thesis on which this article is based.

The research reported in this paper was carried out in the course of the GAMES-II project and the KACTUS project. These projects are partially funded by the Commission of the European Communities. The partners in GAMES-II are SAGO (Italy), Foundation of Research and Technology (Greece), Geneva University Hospital (Switzerland), the University of Amsterdam (The Netherlands), University College of London (United Kingdom), the University of Pavia (Italy) and the University of Ulm (Germany). The partners in the KACTUS project are Integral Solutions Limited (United Kingdom), Labein (Spain), Lloyd's Register

(United Kingdom), Statoil (Norway), Cap Programator (Sweden), University of Amsterdam (The Netherlands), University of Karlsruhe (Germany), Iberdrola (Spain), Delos (Italy), Fincantieri (Italy) and Sintef (Norway).

This article expresses the opinions of the authors and not necessarily those of the consortia.

References

ABEN, M. (1995). *Formal methods in knowledge engineering,* Ph.D. Thesis, University of Amsterdam, Amsterdam, The Netherlands.

ABU-HANNA, A. (1994). *Multiple domain models in diagnostic reasoning.* Ph.D. Thesis, University of Amsterdam, Amsterdam, The Netherlands.

ABU-HANNA, A., BENJAMINS, V. R. & JANSWEIJER, W. N. H. (1991). Functional models in diagnostic reasoning. *Proceedings of The Eleventh International Workshop on Expert Systems and Their Applications, General Conference on Second Generation Expert Systems,* pp. 243–256, Avignon, France.

ALBERT, P. & JACQUES, G. (1993). Putting CommonKADS at work using Kads-Tool. In *Kennis-technologie '93,* Amsterdam, The Netherlands.

ALBERTS, L. K. (1993). *YMIR: an ontology for engineering design.* Ph.D. Thesis, University of Twente,

ANJEWIERDEN, A., SHADBOLT, N. R. & WIELINGA, B. J. (1992*a*). Supporting knowledge acquisition: the Acknowledge project. In *Enhancing the Knowledge Engineering Process—Contributions from ESPRIT,* pp. 143–172. Amsterdam: Elsevier Science.

ANJEWIERDEN, A., WIELEMAKER, J. & TOUSSAINT, C. (1992*b*). Shelley—computer aided knowledge engineering. *Knowledge Acquisition,* **4**.

BARANOV, A., DENSOW, D., FLIEDNER, T. M. & KINDLER, H. (1994). *Clinical Pre-Computer Proforma for the International Computer Database for Radiation Exposure Case Histories.* Heidelberg: Springer.

BENNETT, J. S. (1985). ROGET: a knowledge-based system for acquiring the conceptual structure of a diagnostic expert system. *Journal of Automated Reasoning,* **1,** 49–74.

BOOSE, J. H. (1985). A knowledge acquisition program for expert systems based on personal construct psychology. *International Journal of Man–Machine Studies,* **23,** 495–525.

BOOSE, J. H. & BRADSHAW, J. M. (1988). Expertise transfer and complex problems: using AQUINAS as a knowledge acquisition workbench for knowledge-based systems. In J. H. BOOSE. & B. R. GAINES, Ed. *Knowledge Acquisition For Knowledge Based Systems,* Vol. 2. pp. 39–64. Academic Press.

BRACHMAN, R. J., FIKES, R. E. & LEVESQUE, H. J. (1985). KRYPTON: A functional approach to knowledge representation. In R. J. BRACHMAN. & H. J. LEVESQUE, Eds. *Readings in Knowledge Representation,* pp. 411–429. Los Altos, CA: Morgan Kaufmann.

BREUKER, J. A. & WIELINGA, B. J. (1989). Model driven knowledge acquisition. In P. GUIDA, & G. TASSO. *Topics in the Design of Expert Systems,* pp. 265–296. Amsterdam, North-Holland.

BREUKER, J. A., WIELINGA, B. J., VAN SOMEREN, M., DE HOOG, R., SCHREIBER, A. T., DE GREEF, P., BREDEWEG, B., WIELEMAKER, J., BILLAULT, J. P., DAVOODI, M. & HAYWARD, S. A. (1987). *Model driven knowledge acquisition: interpretation models.* ESPRIT Project P1098 Deliverable D1 (task A1), University of Amsterdam and STL Ltd, Amsterdam, The Netherlands.

BURTON, A. M., SHADBOLT, N. R., RUGG, G. & HEDGECOCK, A. P. (1990). The efficacy of knowledge elicitation techniques: a comparison across domains and levels of expertise. *Knowledge Acquisition,* **2,** 167–178.

BYLANDER, T. & CHANDRASEKARAN, B. (1988). Generic tasks in knowledge-based reasoning: The right level of abstraction for knowledge acquisition. In B. R. GAINES, & J. H. BOOSE, Ed. *Knowledge Acquisition for Knowledge Based Systems,* Vol. 1, p. 65–77. London: Academic Press.

CHANDRASEKARAN, B. (1987). Towards a functional architecture for intelligence based on generic information processing tasks. In *Proceedings of the 10th IJCAI,* pp. 1183–1192, Milan, Italy.

CLANCEY, W. J. & LETSINGER, R. (1984). NEOMYCIN: reconfiguring a rulebased expert system for application to teaching. In W. J. CLANCEY, & E. H. SHORTLIFFE, Eds. *Readings in Medical Artificial Intelligence*: *the First Decade,* pp. 361–381. Reading, MA: Addison-Wesley.

CONSOLE, L., PORTINALE, L., DUPRÉ, D. T. & TORASSO, P. (1993). Combining heuristic reasoning with causal reasoning in diagnostic problem solving. in J. H. DAVID, J. P. & KRIVINE, & R. SIMMONS, Ed. *Second Generation Expert Systems,* p. 46–68 Berlin: Springer-Verlag.

CONSOLE, L. & TORASSO, P. (1988). Heuristic and causal reasoning in CHECK. In *Proceedings of the 12th IMACS World Conference on Scientific Computation88,* pp. 283–286, Paris, France.

DAVIS, R. (1979). Interactive transfer of expertise. *Artificial Intelligence,* **12,** 121–157.

DAVIS, R., SHROBE, H. & SZOLOVITS, P. (1993). What is a knowledge representation? *AI Magazine,* Spring, 17–33.

DE KLEER, J. H. & WILLIAMS, B. C. (1987). Diagnosing multiple faults. *Artificial Intelligence,* **32,** 97–130.

DUDA, R. O., GASCHING, J. G. & HART, P. E. (1979). Model design in the PROSPECTOR consultant system for mineral exploration. In D., MICHIE, Ed. *Expert Systems in the Micro-Electronic Age,* pp. 153–1674. Edinburgh University Press.

ERIKSSON, H., PUERTA, A. R. & MUSEN, M. A. (1994). Generation of knowledge acquisition tools from domain ontologies. In B. R. GAINES, & M. A. MUSEN, Eds. *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop,* pp. 7-1–7-20, Alberta, Canada.

ESHELMAN, L. (1988). MOLE: A knowledge-acquisition tool for cover-and-differentiate systems. In S., MARCUS, Ed. *Automating Knowledge Acquisition for Expert Systems,* pp. 37–80. Boston, MA: Kluwer.

FALASCONI, S. (1993). *Ontological foundations of knowledge based systems in medicine.* Master's thesis, University of Pavia, Italy. (In Italian.)

FALASCONI, S. & STEFANELLI, M. (1994). A library of implemented ontologies. In *Proceedings of the ECAI Workshop on Comparison of Implemented Ontologies,* pp. 81–91, Amsterdam, The Netherlands.

FIKES, R. E. & KEHLER, T. (1985). The role of frame based representation in reasoning. *Communications of the ACM,* **28,** 904–920.

FORD, K. M., BRADSHAW, J. M., ADAMS-WEBBER, J. R. & AGNEW, M. M. (1993). Knowledge acquisition as a constructive modelling activity. *International Journal of Intelligent Systems,* **8,** 9–32.

GRUBER, T. R. (1992). *Ontolingua: a mechanism to support portable ontologies.* Version 3. 0. Technical report, Knowledge Systems Laboratory, Stanford University, CA, U.S.A.

GRUBER, T. R. (1993). A translation approach to portable ontology specifications. *Knowledge Acquisition,* **5,** 199–220.

GRUBER, T. R. (1994). Towards principles for the design of ontologies used for knowledge sharing. In N. GUARINO, & R. POLI. Eds. *Formal Ontology in Conceptual Analysis and Knowledge Representation.* Boston, MA: Kluwer.

GUARINO, N. & BOLDRIN, L. (1993). *Ontological requirements for knowledge sharing.* Paper presented at the IJCAI workshop for knowledge sharing and information interchange, Chambery, France.

KINDLER, H., DENSOW, D. & FLIEDNER, T. M. (1993). A knowledge-based advisor to deal with rare diseases. In *Proceedings AIME'93 Munich, 3–6 October, Medical Artificial Intelligence.* Amsterdam: Elsevier Science Publishers.

KIRSH, D. (1990). When is information explicitly represented. In P. HANSON, Ed. *Vancouver Studies in Cognitive Science 1,* pp. 340–365. Vancouver, BC: University of British Columbia Press.

KLINKER, G., BHOLA, C., DALLEMAGNE, G., MARQUES, D. & MCDERMOTT, J. (1991). Usable and reusable programming constructs. *Knowledge Acquisition,* **3,** 117–136.

LANZOLA, G. & STEFANELLI, M. (1992). A specialized framework for medical knowledge based systems. *Computers and Biomedical Research,* **25,** 351–365.

LENAT, D. B. & GUHA, R. V. (1990). *Building Large Knowledge-Based Systems. Representation and Inference in the Cyc Project.* Reading, MA. Addison-Wesley.

LEVESQUE, H. J. & BRACHMAN, R. J. (1985). A fundamental tradeoff in knowledge representation and reasoning. In R. J. B. H. J. LEVESQUE, Ed. *Readings in Knowledge Representation,* pp. 41–70. San Mateo, CA: Morgan Kaufmann.

LINDBERG, D. A. B., HUMPHREYS, B. L. & MCCRAY, A. T. (1993). The unified medical language system. *Methods of Information in Medicine,* **32,** 281–291.

MACGREGOR, R. (1991). The evolving technologyof classification-based knowledge representation systems. In J. SOWA, *Principles of Semantic Networks: Explorations in the Representation of Knowledge,* pp. 385–400. San Mateo, CA: Morgan Kaufmann.

MAJOR, N. & REICHGELT, H. (1990). ALTO: an automated laddering tool. In B. J., WIELINGA, J. H. BOOSE, B. R. GAINES, A. T. SCHREIBER, & M. VAN SOMEREN, Eds. *Current Trends in Knowledge Acquisition,* pp. 222–236. Amsterdam: IOS Press.

MARCUS, S., Ed. (1988). *Automatic Knowledge Acquisition for Expert Systems.* Boston, MA: Kluwer.

MARCUS, S. & MCDERMOTT, J. (1989). SALT: a knowledge acquisition language for propose-and-revise systems. *Artificial Intelligence,* **39,** 1–38.

MCDERMOTT, J. (1988). Preliminary steps towards a taxonomy of problem-solving methods. In S. MARCUS, Ed. *Automating Knowledge Acquisition for Expert Systems,* pp. 225–255. Boston, MA: Kluwer.

MOTTA, E., RAJAN, T., DOMINGUE, J. & EISENSTADT, M. (1990). Methodological foundations of KEATS, the knowledge engineering assistant. In B. J. WIELINGA, J. H. BOOSE, B. R. GAINES, A. T. SCHREIBER, & M. VAN SOMEREN, Eds. *Current Trends in Knowledge Acquisition,* pp. 257–275. Amsterdam: IOS Press.

MUSEN, M. A. (1989*a*). *Automated Generation of Model-Based Knowledge-Acquisition Tools.* London: Pitman.

MUSEN, M. A. (1989b). Automated support for building and extending expert models. *Machine Learning,* **4,** 347–376.

MUSEN, M. A., FAGAN, L. M., COMBS, D. M. & SHORTLIFFE, E. H. (1988). Use of a domain model to drive an interactive knowledge editing tool. In J. H. BOOSE, & B. R., GAINES, Eds. *Knowledge-Based Systems, Vol. 2: Knowledge Acquisition Tools for Expert Systems,* pp. 257–273. London: Academic Press.

MUSEN, M. A. & SCHREIBER, A. T. (1995). Architectures for intelligent systems based on reusable components. *Artificial Intelligence in Medicine.* Editorial Special Issue.

NECHES, R., FIKES, R. E., FININ, T., GRUBER, T. R., PATIL, R. S., SENATOR, T. & SWARTOUT, W. R. (1991). Enabling technology for knowledge sharing. *AI Magazine,* Fall, 36–56.

NEWELL, A. (1982). The knowledge level. *Artificial Intelligence,* **18,** 87–127.

PATIL, R. S. (1981). *Causal Representation of Patient Illness for Electrolyte and Acid-Base Diagnosis.* Ph.D. Thesis, Laboratory for Computer Science, MIT, U.S.A.

POST, W. M., KOSTER, R. W., ZOCCA, V. & SRAMEK, M. (1993). Cooperative medical problem solving. In *AIME 93–4th Conference on Artificial Intelligence in Medicine Europe,* Munich, Germany.

PUERTA, A. R., EGAR, J., TU, S. W. & MUSEN, M. A. (1992). A multiple-method shell for the automatic generation of knowledge acquisition tools. *Knowledge Acquisition,* **4,** 171–196.

RAMONI, M., STEFANELLI, M., BAROSI, G. & MAGNANI, L. (1992). An epistemological framework for medical knowledge based systems. *IEEE Transactions on Systems, Man and Cybernetics,* **22,** 1361–1375.

RECTOR, A. L., NOWLAN, W. A., KAY, S., GOBLE, C. A. & HOWKINS, T. J. (1993). A framework for modelling the electronic medical record. *Methods of Information in Medicine,* **32,** 109–119.

ROSCH, E. (1973). Natural categories. *Cognitive Psychology,* **4**.

RUNKEL, J. T. & BIRMINGHAM, W. P. (1994). Separation of knowledge: a key to reusability. In B. R. GAINES, & M. A. MUSEN, Eds. *Proceedings of the 8th Banff Knowledge Acquisition for Knowledge-based Systems Workshop,* pp. 36-1–36-19.

SCHREIBER, A. T. (1993). Operationalizing models of expertise. In A. T. SCHREIBER, B. J.

WIELINGA, & J. A. BREUKER, Eds. *KADS: A Principled Approach to Knowledge-Based System Development,* pp. 119–149. London: Academic Press.

SCHREIBER, A. T. & BIRMINGHAM, W. P. (1996). Editorial: the sisyphus VT initiative. *International Journal on Human Computer Studies,* **44,** 275–280.

SCHREIBER, A. T., WIELINGA, B. J., AKKERMANS, J. M., VAN DE VELDE, W. & ANJEWIERDEN, A. (1994). CML: the CommonKADS conceptual modelling language. In L., STEELS, A. T. SCHREIBER, & W. VAN DE VELDE, Eds. *A Future for Knowledge Acquisition. Proceedings of the 8th European Knowledge Acquisition Workshop EKAW'94,* pp. 1–25. Berlin: Springer-Verlag.

SCHREIBER, A. T., WIELINGA, B. J. & JANSWEIJER, W. H. J. (1995). The KACTUS view on the 'O' word. In *IJCAI Workshop on Basic Ontological Issues in Knowledge Sharing,* Montreal, Canada.

SHADBOLT, N. R. & WIELINGA, B. J. (1990). Knowledge based knowledge acquisition: the next generation of support tools. In B. J., WIELINGA, J. H. BOOSE, B. R. GAINES, A. T. SCHREIBER, & M. W. VAN SOMEREN, Eds. *Current Trends in Knowledge Acquisition,* pp. 313–338. Amsterdam: IOS Press.

SHAW, M. L. G. & GAINES, B. R. (1987). An interactive knowledge elicitation technique using personal construct technology. In A. L., KIDD, Ed. *Knowledge Acquisition for Expert Systems: A Practical Handbook.* New York: Plenum Press.

SHAW, M. L. G. & GAINES, B. R. (1989). Comparing conceptual structures: consensus, conflict correspondence and contrast. *Knowledge Acquisition,* **4,** 341–364.

SHORTLIFFE, E. H. (1979). *Computer-Based Medical Consultations: Mycin.* New York: American-Elsevier.

SIMMONS, R. (1992). The roles of associational and causal reasoning in problem solving. *Artificial Intelligence,* **53,** 159–208.

SIMMONS, R. (1993). Generate test and debug: a paradigm for combining associational and causal reasoning. In J. M. DAVID, J. P. KRIVINE, & R. SIMMONS, Eds. *Second Generation Expert Systems,* pp. 79–92. Berlin: Springer-Verlag.

SRINIVAS, S. & BREESE, J. (1990). Ideal: a software package for analysis of influence diagrams. In *Proceedings of 6th Conference on Uncertainty in AI,* Cambridge, MA, U.S.A.

STEELS, L. (1985). Second generation expert systems. *FGCS,* **1,** 213–221.

STEELS, L. (1990). Components of expertise. *AI Magazine,* **11,** 30–49.

STEELS, L. (1993). The componential framework and its role in reusability. In J. M. DAVID, J. P. KRIVINE, & R. SIMMONS, Eds. *Second Generation Expert Systems,* p. 273–298. Berlin: Springer-Verlag.

TU, S. W., ERIKSSON, H., GENNARI, J. H., SHAHAR, Y. & 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,** 257–289.

VAN HEIJST, G., TERPSTRA, P., WIELINGA, B. J. & SHADBOLT, N. R. (1992). Using generalized directive models in knowledge acquisition. In T. WETTER, K. D. ALTHOFF, J. H. BOOSE, B. R. GAINES, M. LINSTER, & F. SCHMALHOFER, Eds. *Current Developments in Knowledge Acquisition: EKAW-92,* pp. 112–132, Berlin: Springer-Verlag.

VAN MELLE, W. (1979). A domain independent production rule system for consultation programs. In *IJCAI-79,* pp. 923–925, Tokyo, Japan.

WEISS, S. M. & KULIKOWSKI, C. A. (1979). EXPERT: a system for developing consultation modles. In *Proceedings of IJCAI,* pp. 826–832.

WEISS, S. M., KULIKOWSKI, C. A., AMAREL, S. & SAFIR, A. (1984). A model-based method for computer-aided medical decision making. In W. J. CLANCEY, & E. H., SHORTLIFFE, Eds. *Readings in Medical Artificial Intelligence, the First Decade.* Reading, MA: Addison Wesley.

WIELEMAKER, J. & ANJEWIERDEN, A. (1989). Separating user interface and functionality using a frame based data model. In *Proceedings Second Annual Symposium on User Interface Software and Technology,* pp. 25–33. Williamsburg, VA: ACM Press.

WIELINGA, B. J. & BREUKER, J. A. (1986). Models of expertise. In *Proceedings ECAI-86,* pp. 306–318, Brighton, UK.

WIELINGA, B. J. & SCHREIBER, A. T. (1993). Reusable and sharable knowledge bases: a European perspective. In *Proceedings International Conference on Building and Sharing of Very Large-Scaled Knowledge Bases,* pp. 103–115. Tokyo, Japan: Japan Information Processing Development Center.

WIELINGA, B. J., SCHREIBER, A. T. & BREUKER, J. A. (1992). KADS: a modelling approach to knowledge engineering. *Knowledge Acquisition,* **4** 5–53. Reprinted in BUCHANAN, B. & WILKINS, D. Ed. (1992). *Readings in Knowledge Acquisition and Learning,* pp. 92–116. San Mateo, CA: Morgan Kaufmann.

WIELINGA, B. J., VAN DE VELDE, W., SCHREIBER, A. T. & AKKERMANS, J. M. (1993). Towards a unification of knowledge modelling approaches. In J. M. DAVID, J. P. KRIVINE, & R. SIMMONS, Eds. *Second Generation Expert Systems,* pp. 299–335. Berlin: Germany, Springer-Verlag.

YOST, G., KLINKER, G., LINSTER, M., MARQUES, D. & MCDERMOTT, J. (1994). The SBF framework, 1989–1994: from applications to workplaces. In L. STEELS, W. VAN DER VELDE, & A. T. SCHREIBER, Eds. *Proceedings European Knowledge Acquisition Workshop EKAW*'94. Berlin: Springer Verlag.