Building a KBS for Health and Safety Assessment [1]

John Kingston

AIAI-TR-202

Artificial Intelligence Applications Institute,
University of Edinburgh,
80 South Bridge,
Edinburgh, EH1 1HN
United Kingdom

Abstract

EASE ('Estimation and Assessment of Substance Exposure') is a knowledge-based system for assessing workplace exposure to potentially hazardous new substances. It was built for the Health and Safety Executive of the United Kingdom (HSE). EU regulations require a manufacturer of a new substance to notify the appropriate authority, who will carry out a risk assessment of it. To aid this process, guidance is provided by the regulator for use by both authorities and manufacturers. This is usually provided as a paper document, but HSE decided that a computer based guidance system would be of benefit. Safety-related considerations and a desire for quality led to the system being developed in accordance with ISO9001 standards.

The system guides the user by offering a menu of appropriate choices whenever it needs information. The use of CommonKADS ensured that the problem requirements and the expert knowledge involved were captured within a standard framework which promoted unambiguous communication between the members of the project team and provided a solid base for system design, implementation, maintenance, and enhancements.

The system was implemented using the NASA CLIPS development tool for the inference engine and knowledge base. Initially required to run under MS-DOS on a PC AT equivalent with 640K of RAM, a second release to run under Windows 3.1 reused the inference engine and knowledge base, requiring only a revised user interface.

The system has been widely distributed for use by authorities throughout Europe.

---

[1]Also appears in *Applications and Innovations in Expert Systems IV, Proceedings of BCS Expert Systems '96, Cambridge, 16-18 December 1996. SGES Publications.*

1 Problem Description

EASE ('Estimation and Assessment of Substance Exposure') is a knowledge-based system for assessing workplace exposure to potentially hazardous substances, which was built for the Health and Safety Executive of the United Kingdom. EU directives and regulations require manufacturers or suppliers of new or existing substances to notify the appropriate authority, who will carry out a risk assessment of it. To aid this process, guidance is provided by the regulator for use by both authorities and manufacturers. This is provided as a paper document, but HSE decided that a computer based guidance system would also be of benefit.

The objective of the application is to enable assessors to produce a reliable estimate of the degree to which workers will be exposed to the new or existing substance. Exposure assessment is an expert task, and is best performed by an experienced occupational hygienist. For the system to be deployed, it must incorporate best practice knowledge engineering techniques and, as there are safety-related considerations and both HSE and the developers of the system are committed to quality, the work had to be carried out in accordance with ISO9001 standards.

A knowledge-based system was built which ensures that assessors remember all the necessary questions to ask when estimating exposure to a substance, and then calculates the exposure level. The system also incorporates backtracking facilities in order to allow errors to be corrected, and 'What-If' analyses to be performed. Two versions of the system were built. The first was targeted to run on 286 PCs (or higher) under MS-DOS, which was the only hardware available to some manufacturers when the system was built; a subsequent version was designed to run under Windows 3.1 with a revised user interface. The system had been designed in a modular fashion, so that the second version of the system required only a few changes to the inference engine and knowledge base, and most of these changes were in response to users' comments.

2 Knowledge Acquisition and Analysis

A prototype of this KBS had previously been produced using a different KBS programming tool. The textual knowledge base from the prototype of the KBS was used as a transcript, which was analysed in the same way as any interview transcript: fragments of text were identified as being relevant to the task of problem solving, and were stored in dictionaries of concepts, properties, tasks, or other ontological types. In addition, a number of decision trees were built (e.g. Figure 1) which represented some of the decision processes which take place during problem solving. These decision trees, along with the knowledge analysis models which are described later, were produced using TOPKAT, which is a tool for supporting knowledge acquisition and knowledge modelling [Kin94].

The knowledge analysis models which were developed used the CommonKADS methodology [Bv94]. CommonKADS is the name of the methodology developed by the KADS-II project, which was funded under the CEC ESPRIT programme [SWd+94] [Wie93].

CommonKADS views KBS development as a modelling process. Knowledge analysis is performed by creating up to six models which represent the knowledge from different viewpoints. CommonKADS recommends a number of different models, which start with the representation of various aspects of an organisation, and support the whole knowledge engineering process up to the point of producing a detailed design specification.

Four of the six CommonKADS models are primarily concerned with identification of a suitable application for an AI system, and so were not considered relevant for this application. This was because the application for the KBS had already been chosen, and because an earlier attempt at automation had resolved many of the issues connected with the proposed role of the knowledge based system, and its required inputs and outputs. However, the remaining two models (the Expertise Model and the Design Model) were built.

A member of HSE's staff cooperated in the development of the models, allowing him to understand the models sufficiently to criticise them constructively, and to modify them in the future. A few questions were referred back to a senior expert at HSE, who also provided staff to perform user testing of the system.

3 Knowledge Analysis: CommonKADS Expertise Model

The CommonKADS Expertise model is divided into three 'levels' representing different viewpoints on the expert knowledge:

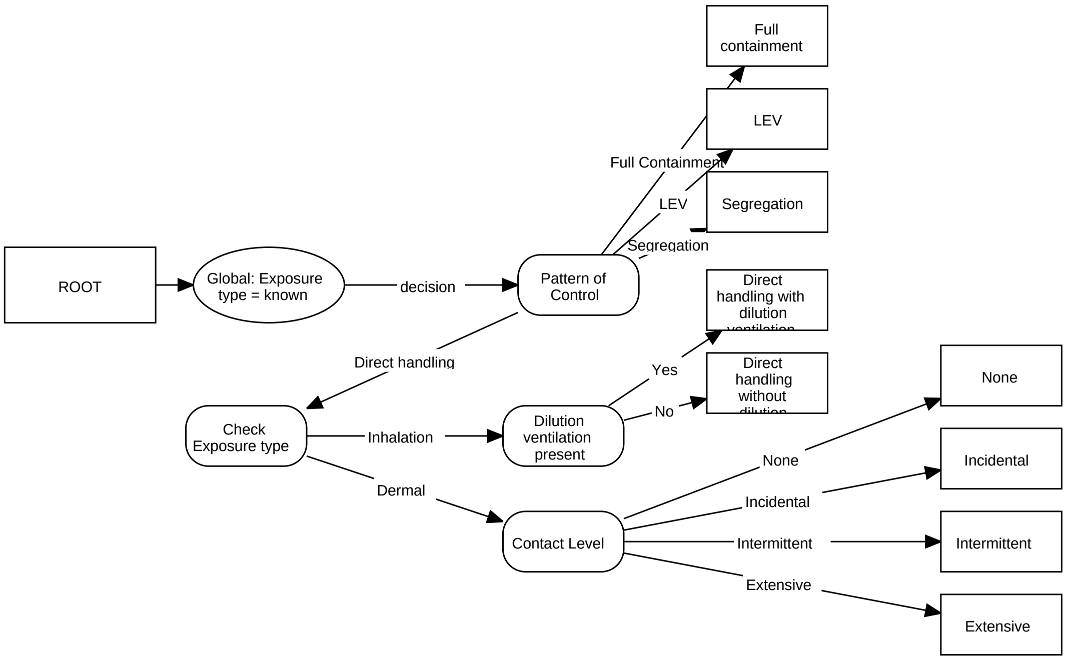

Fig 1: Decision tree: ask pattern of control

- The **domain knowledge** which represents the declarative knowledge in the knowledge base;
- The **inference knowledge** which represents the knowledge-based inferences which are performed during problem solving;
- The **task knowledge** which defines a procedural ordering on the inferences.

The contents of these three levels can be defined graphically, or using CommonKADS' Conceptual Modelling Language.

3.1 Domain knowledge

CommonKADS recommends that acquired knowledge is classified into one of the following ontological categories:

- Concept;
- Property;
- Expression;
- Relation.

Tasks which are performed as a necessary part of problem solving may also be identified.

The process of transcript analysis generated 83 concepts, 31 properties, 8 expressions, 4 tasks, and no relations (although some relations were added later). The concepts were further subdivided into the following six categories:

- Chemical compounds – different chemical compounds
- Exposure types – different ways in which exposure to a substance can occur
- Patterns of control – ways of reducing exposure to a substance
- Patterns of use – ways in which the substance is used, handled, etc.
- Physical states – possible physical states of a substance
- Vapour pressure values – the vapour pressure value can be measured, calculated or estimated.

Two subcategories of properties were also identified:

- Substance properties – properties of the substance;
- Process properties – properties relating to the process.

3.2 Domain models

Domain models show relationships between domain dictionary items. Each domain model usually shows all instances of one type of relation (e.g. causal relations). When the type of relation is *is-a* or *instance-of*, then the domain model forms a taxonomic hierarchy.

Six domain models were defined on this project. They are:

- Exposures to check – shows which physical states *may cause* certain types of exposure;
- Hierarchy of substances – a taxonomic hierarchy of chemical compounds;
- System estimation of vapour pressure & Vapour pressure extrapolation – two models showing *input/output* relationships between tasks & (numerical) concepts in complex calculations;
- Types of exposure – a simple taxonomic hierarchy of exposure types;
- Use / control incompatibilities – defines patterns of use which are *incompatible with* certain patterns of control.

An example of a domain model can be seen in Figure 2.

3.3 Model schemata

A model schema is a statement which describes the content of a domain model at an abstract level e.g. **physical states** *may cause* **exposure types**. Model schemata therefore define relations between abstracted domain terms. The purpose of the model schemata is to provide a bridge between the domain level and the inference level of the model of expertise, and to produce a model of the domain which is sufficiently abstract to be re-used in other KBS projects in the same domain.

A diagram encompassing all defined model schemata can be seen in Figure 3.

3.4 Model of Expertise: Inference level

The inference level of the model of expertise represents the inference processes which are performed in order to solve the task in hand. A model is constructed which shows *inference steps* (decisions which are taken or inferences which are drawn during problem solving) and *knowledge roles* (the types of knowledge which form input and output to the inference steps).

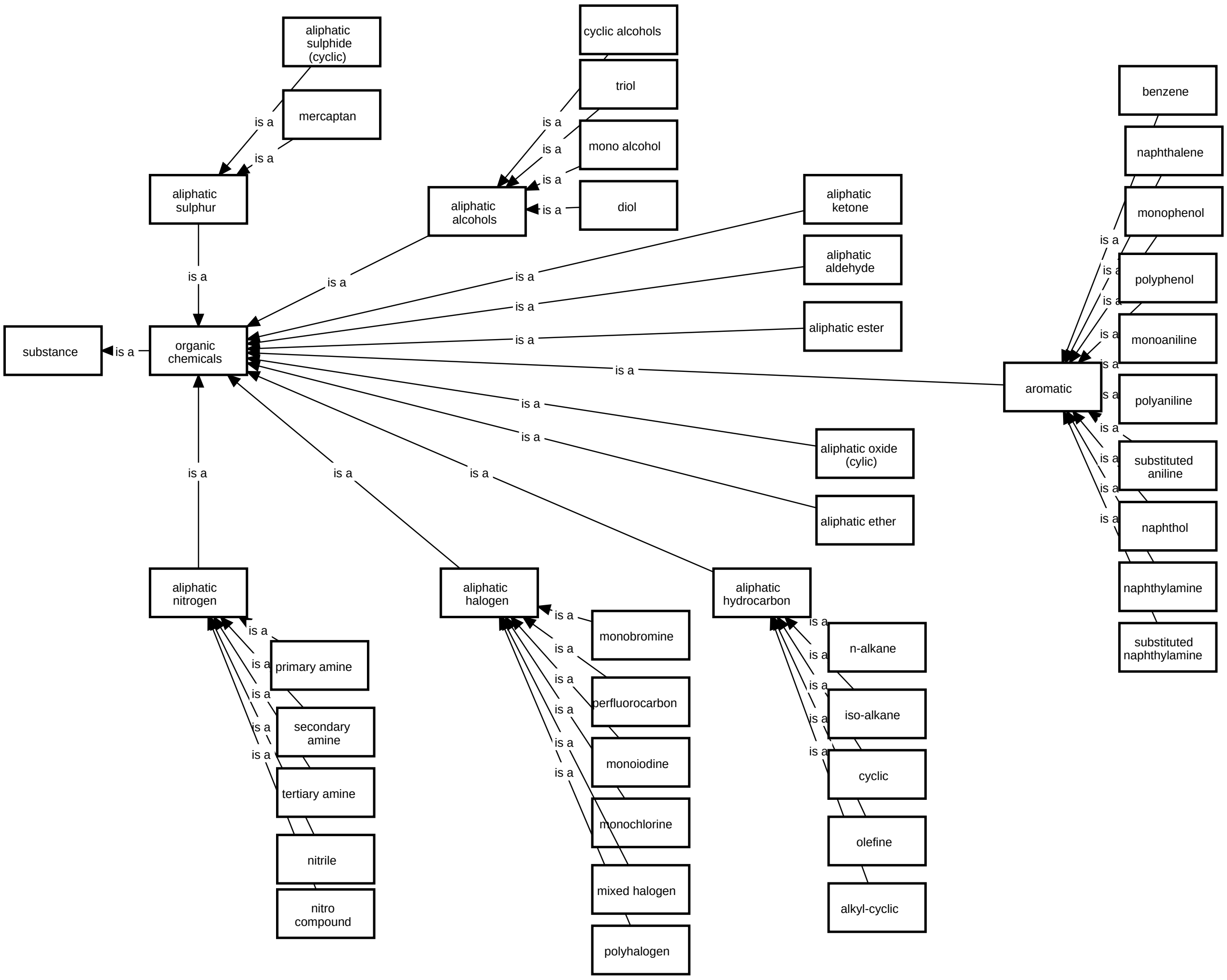

Fig 2: Domain model: Hierarchy of substances

The inference level is normally constructed by identifying the type of task which is being tackled, and then selecting a generic inference structure for that task type from CommonKADS' library of generic inference structures. In this case, it was determined that the task type was **Assessment**. For Assessment tasks, a further level of refinement is available: [LV93] have published a paper which shows how the generic inference structure for Assessment tasks can be configured to the task in hand, in order to reflect the actual inferences which are performed more closely than a single generic inference structure would.

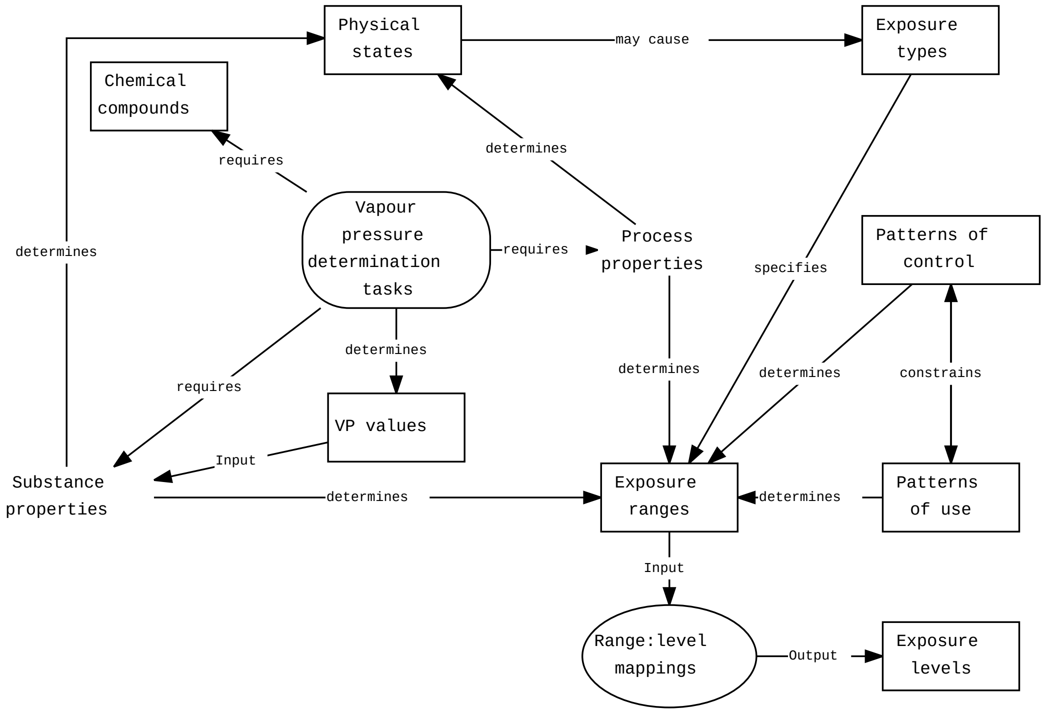

Fig 3: Model schemata

3.5 Configuring the generic inference structure

The basic inference structure for Assessment tasks is shown in Figure 4. The 'case description' was initially identified with the substance, and with the process in which the substance was being used; the 'system model' was initially identified with the contents of an occupational hygiene database; and the 'decision class' was initially identified as being one of a number of exposure ranges (e.g. *0-0.1* mg/m^3). On the basis of these identifications, a set of questions was answered; these questions guide configuration of the inference structure.

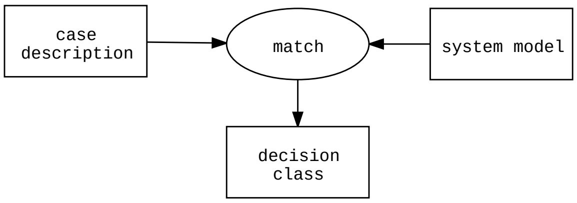

Fig 4: Basic inference structure for Assessment tasks

Firstly, the questions determine if there is need for any configuration at all. If the case description and the system model are in a similar format at a similar level of abstraction, so that they can be matched directly, then no changes to the inference structure are needed. However, in this case the substance and process cannot be matched directly against an occupational hygiene database, because the substance is a new substance, and the database does not store full descriptions of processes; instead, both the substance description and the contents of the database must be transformed in some way to allow matching. The configuration guidance suggests that the case description will need to be *abstracted* (i.e. key pieces of information will need to be selected) while the system model will need to be *specified* (i.e. some information, or structuring of information, will need to be added to it). Figure 5 shows the inference structure resulting from these changes.

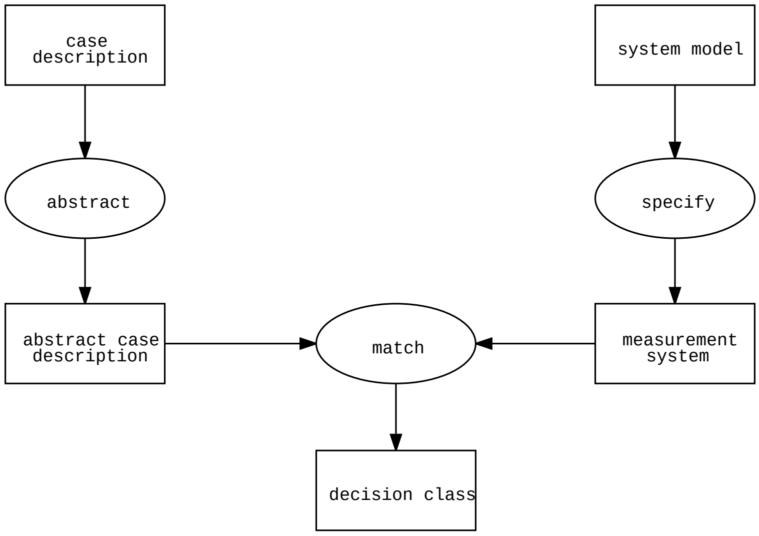

Fig 5: Inference structure after initial configuration.

Having determined that abstraction and specification are required, further questions are asked to determine if these steps need to be refined to an even greater level of detail. The key question for the abstraction step was whether the case description was already adequate (in content or structure) to be abstracted. It seemed that while the available information about the substance and the process were adequate for abstraction, the information that was actually used was dependent on the type of exposure which was being investigated (exposure to inhaled dust, exposure to inhaled vapour, or dermal exposure). A *select* inference step was therefore added to the configured inference structure, with input from a *filter* knowledge role. As for the specification step, the first question asks if the system model needs to be focused because there is more than one type of system (not true) and if the measurement system is independent of the case description (true). These answers normally specify no changes; however, the measurement system is affected by the type of exposure being investigated, so a single inference step is added to **specify** the measurement system to be used, based on the *filter* knowledge role (i.e. the chosen exposure type).

There are also a set of questions available for configuring the matching process; however, the answer to the first question (Is the decision class the direct result of matching the case against the measurement system?) is YES, and so no changes need to be made to the inference structure, and no further questions need to be asked.

The resulting configured inference structure is shown in Figure 6.

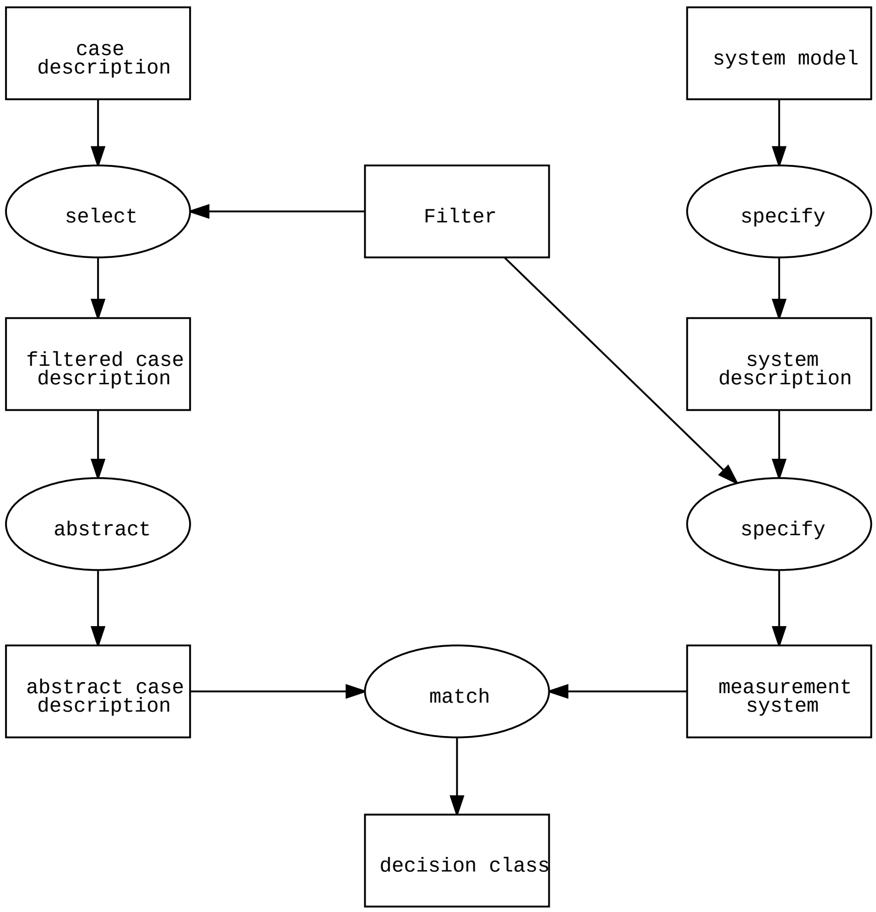

Fig 6: Configured inference structure

3.6 Instantiating the configured inference structure

The instantiated version of the configured inference structure is shown in Figure 7. The case description has been instantiated into **Substance** and **Process**, the measurement system has been instantiated into **Mappings for exposure type**, and the decision class has been instantiated into **Exposure ranges**. Many of the instantiations have been made by consulting the model schemata diagram (Figure 3), and relating abstract concepts from that diagram to knowledge roles in the inference structure.

The instantiated inference structure has some differences from the configured generic inference structure. These differences can be accounted for as follows:

- The comparison with the model schema highlights the fact that some domain-specific knowledge roles are not represented in the inference structure. A good example is that the exposure ranges which are obtained from the **match**ing process are finally converted into an exposure level. In this inference structure, **match-3** produces an **Interpretation**, which maps to the use of *Range–Level mappings* to produce *Exposure Levels* in the model schemata diagram.
- The **system model** has disappeared from the instantiated inference structure. The reason for this is that the **system model** had been provisionally matched to the contents of an occupational hygiene database, which was expected to specify mappings of properties of the substance & process to exposure levels. This activity is performed, but it is 'pre-compiled'; i.e. the mappings have been computed independently from the KBS, and the KBS only needs to use these mappings as an input. The instantiated inference structure therefore does not show the process of specifying these mappings; instead, the mappings are displayed as a *static knowledge role* i.e. knowledge which never alters during a run of the KBS. Static knowledge roles are indicated in the diagram by bold borders and bold arrows.

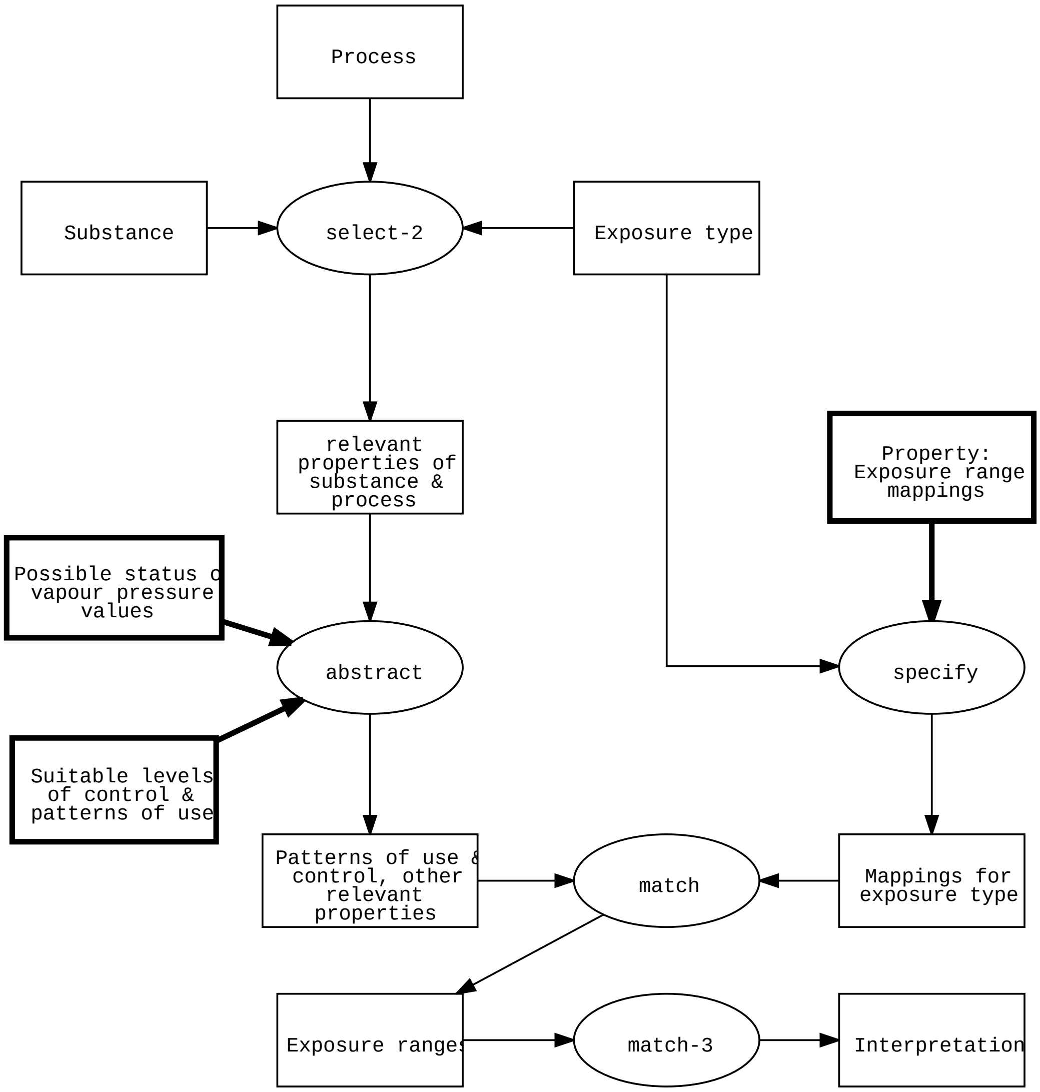

Fig 7: Instantiated inference structure (top level)

3.7 Model of Expertise: Task Level

The task level of the model of expertise extracts all the inference steps and transfer tasks from the various instantiated inference structures, and specifies the control which operates on them. This primarily consists of defining an ordering on the tasks, although it may also include control operators such as **repeat ... until** or logical operators such as **or**.

The task structure for the complete EASE system is shown in Figure 8. The inference steps from Figure 7 can be seen in the leftmost column; the remaining tasks represent inference steps which specify the performance of certain top level inference steps at a greater level of detail.

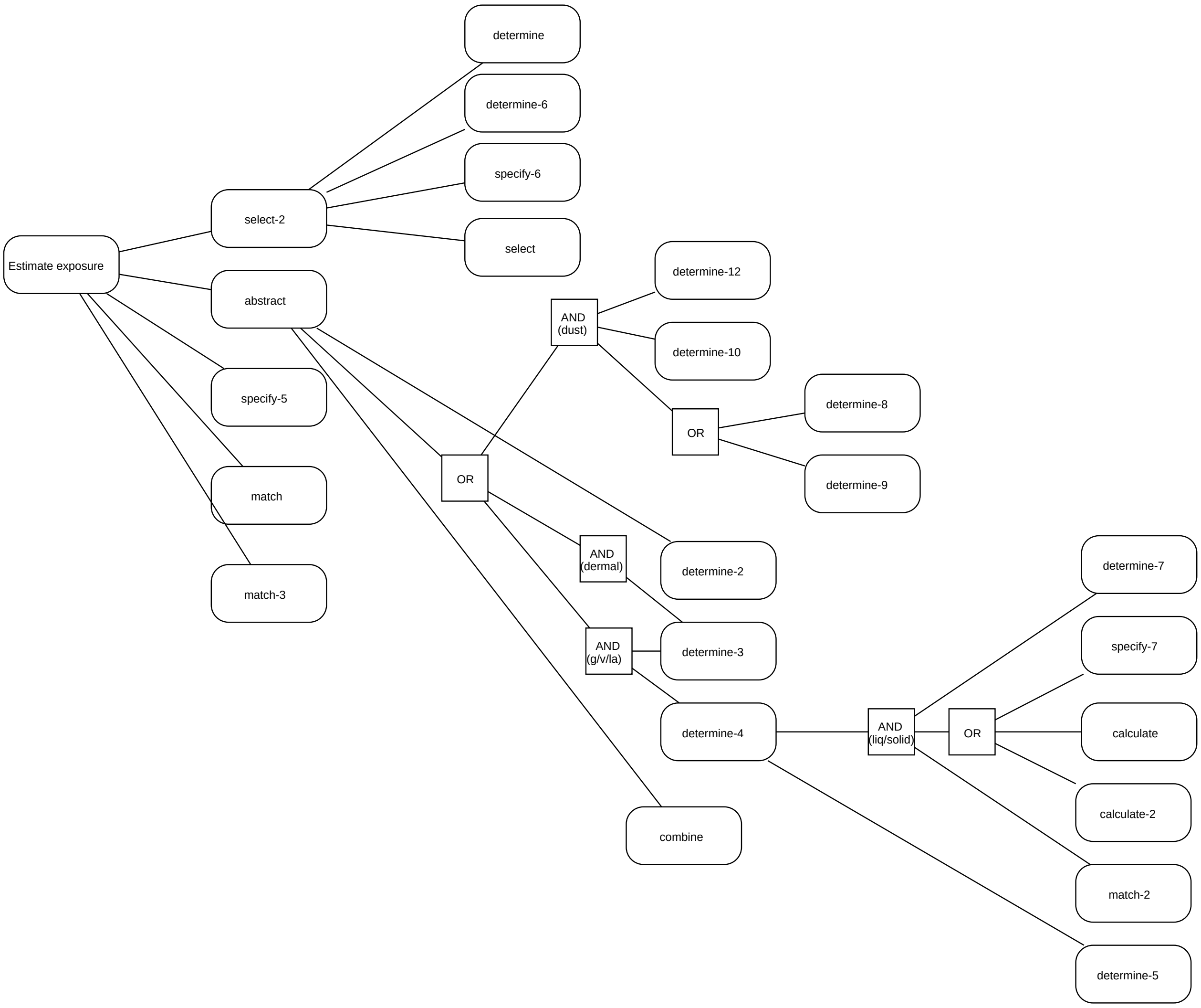

Fig 8: Task structure

3.8 Model of Expertise: Detailed Communication Model

The CommonKADS Expertise Model is comprehensive in its coverage of the knowledge required for problem solving. However, it does not have a model for representing the inputs and outputs required during problem solving. This is assumed to be represented in the Communication Model, which builds on the Task Model by linking required communication with particular tasks. In practice, it is more helpful to develop a 'detailed communication model' which builds on the Task Structure instead of the Task Model.

It uses the format of the CommonKADS Communication Model: that is, agents (those who interact with the KBS – usually a user) and ingredients (items which are output & inputs which are received).

The top level diagram of the detailed communication model can be seen in Figure 9.

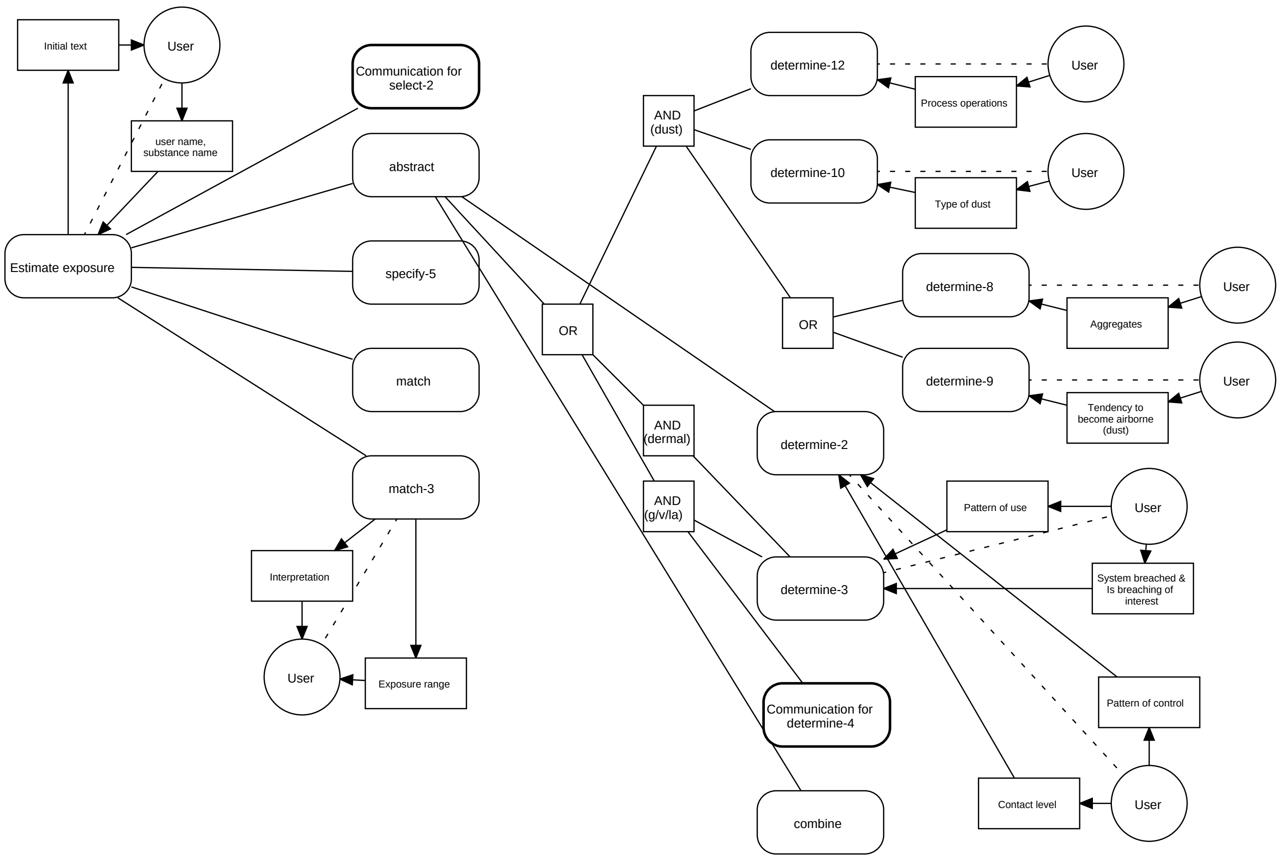

Fig 9: Detailed communication model (top level)

4 KBS Design

4.1 Application Design

Once the model of expertise has been completed, the CommonKADS design model can be developed. The development of the design model breaks down into three phases:

- Application design: a high-level decomposition of the model of expertise into sub-systems and the conceptual decomposition of each of these sub-systems;

- Architecture design: focuses on the high-level computational infrastructure within which the application design will be realised;
- Platform design: the lower level decisions about target language, hardware and software configuration, user and user environment.

Application Specification

The application design is the first of these three phases. It breaks down into two stages: the *application specification* and the *detailed application design*. The application specification performs the high-level decomposition of the model of expertise into sub-systems. This corresponds to a functional decomposition of the model of expertise. The report by [Sch93], showing an application of the CommonKADS design model, achieves this using a diagram which recreates the task structure from the model of expertise, and then adds the inference step linked to each task, the knowledge roles linked to each inference step, and the domain level categories linked to each knowledge role. In other words, this diagram captures the whole of the model of expertise in a single diagram. For this project, it was decided that the task structure provided a sufficiently detailed functional decomposition of the problem to be tackled, and so the application specification diagram was not needed.

Detailed application design

The detailed application design takes each primitive task (i.e. every leaf node of the task structure) and performs a 'conceptual decomposition' of that task; that is, an *architecture command* is defined which will fulfil the functionality of that task. Typical architecture commands might be `sub_set(:set options :key exp_type :set relevant_options)` or `get_prop(:concept subst :property b_pt)`. For example, the *sub_set* command fulfils part of the functionality of *specify-5*, which picks out the **mappings** relevant to the chosen **exposure type**. Architecture commands are also produced for the Input/Output transactions defined in the detailed communication model.

Once all the architectural commands had been identified, they were compiled into the following list:

- `ask_choice(:set :prop_or_conc)` – ask the user to choose one value from a choice of many;
- `ask_val(:concept :property)` – ask the user (or another external source) to provide a value;
- `ask_boolean(:concept :property)` – ask the user (or another external source) to provide a Yes/No value;
- `copy_val(:concept :concept)` – copy a value from one data structure to another;

- display_text(:concept :optional :concept) – display text to the user;
- get_el(:set :key :el) – get one value from a set of values (in this case, get a table from a set of tables);
- get_prop(:concept :property) – get the value of a property;
- match_1(:concept :concept) – given a value, look up an equivalent value on a different scale;
- match_2(:key :key :prop_or_conc) – look up a value in a 2-dimensional table, given the value on the 2 dimensions;
- match_3_4(:property :property :property :property :concept) – look up a value in a N-dimensional table (where N is greater than 2), given the appropriate value on each dimension;
- math_N(:property etc) – any purely mathematical operation; item sub_set(:set :key :set) – given a set of values, produce a sub-set differentiated by the key provided.

4.2 Architectural Design

The task of architectural design is to define a computational infrastructure capable of implementing all the architecture commands defined in the detailed application design. Architectural design is also divided into two stages: the *architecture specification* and the *detailed architecture design*.

Architectural Specification

The architecture specification describes the abstract machinery that is used to implement the functionalities of architecture commands. This means deciding whether the architecture is to be rule-based, object-oriented, procedural, or whether a particular AI programming paradigm needs to be used (e.g. blackboard reasoning or model-based reasoning). The system developers use a set of 'probing questions' which can assist the process of deciding if a particular architecture is appropriate. These questions are designed to encode a developer's heuristics about the task (see [Mac92] or [KD89]). The recommendations of the probing questions were:

- Use rule-based programming [**+8**];
- Use data-driven reasoning [**+6**];
- Use shallow reasoning [**+3**];
- Use goal-driven reasoning [**+3**];

- Don't use confirmation by exclusion [**-5**].
- Use canned text for explanations [**+4**];
- Use objects to represent substances & processes [**+4**];
- Use facts [**+2**] or objects [**+2**] to represent tabular information.

This is a fairly consistent set of recommendations: shallow reasoning can be implemented relatively simply using rule-based programming, and both data-driven and goal-driven reasoning can be implemented using rules. The recommendations for objects suggest a system in which rules can pattern match on objects, rather than facts alone.

Detailed architecture design

The next step is the detailed architecture design, which realises the means-end relationship between an architecture design and a target language. It does this by choosing programming techniques appropriate to the architecture specification to implement each of the architecture commands.

The chosen mappings are shown below:

- `ask_choice(:set :prop_or_conc)` – Output a *multiple-selection menu* which returns the selected value;
- `ask_boolean(:concept :property)` – Output a *multiple-selection menu* containing the values YES and NO which returns the selected value;
- `ask_val(:concept :property)` – Output *canned text*, *read* a value;
- `copy_val(:concept :concept)` – possibly a *rule*, possibly a simple *copy* command;
- `display_text(:concept :optional :concept)` – use *canned text*;
- `get_el(:set :key :el)` – This command is highly dependent on the nature of the key, and may need more than one subroutine to implement it. Typically, however, the key will be a logical operator (e.g. element *greater than or equals* value), and so iterated matching of the element with each member of the set will be required. Given the recommendations of the architecture specification, it is highly likely that this iterated matching will be performed by defining a *rule* to perform the matching process;
- `get_prop(:concept :property)` – a simple *get-attribute* command on an object should suffice;
- `match_1(:concept :concept)` – The implementation of this command will depend on how the mappings are defined. It may require *table lookup*, or it may be as simple as a *get-attribute* command;

- `match_2(:key :key :prop_or_conc)` – this could be implemented using *table lookup*, but is more likely to be handled using *rules*;
- `match_3_4(:property :property :property :property :concept)` – this is a prime candidate for implementation using *rules*;
- `math_N(:property etc)` – it is assumed that the chosen language provides basic mathematical functionality;
- `sub_set(:set :key :set)` – This command is highly dependent on the nature of the key, and may need more than one subroutine to implement it. However, the key has to return TRUE or FALSE, and so is likely to be a logical operator of some kind. Therefore, the recommended programming techniques are the same as for `get_el`: *rules* to perform an iterated matching process.

4.3 Platform design

Only now does CommonKADS design consider the platform on which the design will be implemented, although in reality, a wise designer will have ensured that most or all of the recommended architecture specification can be implemented on the chosen platfom while performing detailed architectural design. In this case, the specified implementation vehicle (for the first version of EASE) was CLIPS, running under DOS on PCs with a minimum of a 286 MHz processor. The requirement to run under DOS (rather than Microsoft Windows) imposed a memory restriction, for CLIPS is unable to make use of extended memory in the absence of Microsoft Windows unless it is compiled with overlays, which slow the system down considerably. The preferred solution to this problem was to compile out some of the facilities of CLIPS, leaving it with only the bare essentials needed for the project.

The version of CLIPS which was used to implement this system therefore contained

- forward chaining rules;
- facts;
- I/O functions;
- arithmetic capabilities.

A simple windowing system for I/O, which was implemented specifically for version 1 of EASE, was also compiled in with CLIPS. Objects were *not* included; this was not a big problem, because neither inheritance nor object-oriented programming are required. CLIPS can use a collection of facts with the same initial symbol instead of objects: for example, the substance was represented by a series of facts such as:

```
(substance name water)
(substance melting_point 0)
(substance boiling_point 100)
etc.
```

For version 2 of EASE, a full version of CLIPS was used, the collections of facts which represented properties of substances and processes were replaced with objects, and the simple windowing system was replaced with a more complex user interface based on the freely available WXWINDOWS GUI development package [Sma93].

The platform design for the EASE system followed the recommendations for the detailed architectural design, with the exception of `get_prop` and `match_1` ; as there are no objects, and therefore no *get-attribute* commands, these commands were implemented by using *rules*.

In summary, it can be seen that the restrictions of the platform required a small number of changes to the detailed architectural design. The necessity to make changes which are (presumably) deviations from the optimal design is balanced against the cost of changing the platform. In this case, the deviations are small, and the cost of changing the platform is very high, so the changes are acceptable.

5 Implementation

The implementation of the EASE system was guided by the following principles:

1. The implementation should resemble the design as closely as possible, since the design has been produced by careful analysis and repeated transformations;
2. The implementation should separate declarative and procedural knowledge as far as possible. Declarative knowledge (such as concepts, properties, text, etc) is stored as data which is used by generic rules, rather than being hard-coded into particular rules. This separation greatly simplifies maintenance of the declarative knowledge; however, it may have a small negative effect on the efficiency of the system;
3. The implementation should avoid placing an excessive load on the system's memory. This is because the target delivery vehicle for version 1 of EASE was a 286 PC without Microsoft Windows; on such a machine, CLIPS can only make use of the basic 640K of RAM;
4. The implementation should make use of as few CLIPS constructs as possible, because of the memory restrictions described above.

The resulting structure of the implementation was as follows:

- Declarative knowledge was implemented using facts;

- Each inference step in the design was represented by a set of rules. These rule sets are implemented as separate files (one file per inference step);
- The I/O functionality specified in the detailed communication model was implemented using a small set of rules, which are triggered by facts that are created when needed. These functions correspond to the architectural commands *get_el* and *get_prop*;
- Control of the system is handled by rule-based processing, using logical dependency to ensure that the knowledge base remains consistent.

6 Application Building

The project was carried out by one member of development staff and one member of HSE staff, with occasional assistance from three other members of development staff (primarily on development of user interfaces). Both the initial project and the revisions for a Windows environment took about 50 man days of development staff time. HSE copied the system onto diskettes and the EU distributed it to users in regulatory authorities throughout Europe, using the same distribution channels through which they had formerly sent paper copies of regulations. The member of HSE staff who helped to build the system should be capable of maintaining the system; however, a future project has been planned which will allow the system to take input from a database, thus transferring some of the responsibility for maintenance to the maintainer of the database.

The safety-related nature of this project led to both development projects being audited for compliance with ISO9001 quality standards. Staff from Lloyd's Register of Shipping made frequent visits to ensure the compliance of the project with the standards of ISO9001, and copious documentation was produced to record each stage of the project for subsequent auditing.

The application was evaluated by distribution of a prototype to potential users during development, and by inviting manufacturers and regulatory authorities to provide feedback on the delivered system. User feedback on the application has been positive; some suggestions for improvements on version 1 of EASE have been incorporated into version 2. Perhaps the major benefit of the project to the HSE has been the likely standardisation of European regulatory authorities on the British approach to exposure assessment, because the approach has been encapsulated in software which is widely available. Further projects are planned to make EASE available under a different operating system, and to extend its capabilities to integrate with an occupational hygiene database as well as other software packages.

References

[Bv94] J. Breuker and W. van de Velde. *The CommonKADS Library: reusable components for artificial problem solving.* IOS Press, Amsterdam, Tokyo, 1994.

[KD89] P. J. Kline and S. B. Dolins. *Designing expert systems : a guide to selecting implementation techniques*. Wiley, 1989.

[Kin94] J.K.C. Kingston. Linking Knowledge Acquisition with CommonKADS Knowledge Representation. In *Expert Systems 94*. British Computer Society, Cambridge University Press, Dec 1994. Also available as AIAI-TR-157.

[LV93] C. Löckenhoff and A. Valente. A Library of Assessment Modelling Components. In *Proceedings of 3rd European KADS User Group Meeting*, pages 289–303. Siemens, Munich, March 1993.

[Mac92] C MacNee. PDQ: A knowledge-based system to help knowledge-based system designers to select knowledge representation and inference techniques. Master's thesis, Dept of Artificial Intelligence, University of Edinburgh, September 1992.

[Sch93] R. Schrooten. Sabena flight schedule case: an example of a design model. CommonKADS Deliverable D.M.7 KADS-II/M7/VUB/RR/064/2.1, Vrije Universiteit Brussel, 1993. This report has been included in the CommonKADS 'Design Model and Process' report, the number of which is given above.

[Sma93] J. Smart. HARDY. *Airing*, 15:3–7, April 1993. AIAI, University of Edinburgh.

[SWd+94] G. Schreiber, B. Wielinga, R. de Hoog, H. Akkermans, and W. van de Velde. CommonKADS: A Comprehensive Methodology for KBS Development. *IEEE Expert*, pages 28–37, Dec 1994.

[Wie93] B. Wielinga. Expertise Model: Model Definition Document. CommonKADS Project Report, University of Amsterdam, October 1993. KADS-II/M2/UvA/026/2.0.