

A Purpose Driven Method for Language Comparison

The REVISE project^{12*}

¹ SWI, University of Amsterdam Roetersstraat 15, 1018 WB Amsterdam, The Netherlands

² Faculty of Mathematics and Computer Science Vrije Universiteit De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

Abstract. Current efforts to compare knowledge engineering (KE) modelling languages have been limited to either rather shallow comparisons on a broad-set of languages, or to detailed comparisons with limited applicability to a narrow set of languages. In this paper we propose a novel way of organising language comparisons. This method is based on an alternating decomposition of the goals that a language tries to achieve and the linguistic methods it employs to achieve these goals. This new method for comparing languages allows a general comparison at high levels of abstraction, while not preventing more precise comparisons whenever possible. One result of our comparison method is an insight in the different assumptions that underly the languages to be compared. Two further consequences follow from the proposed comparison method, namely (i) a measure for the degree of similarity between languages, and (ii) a method for translating between languages. After describing our method, we apply it to a pair of KE modelling languages, and show how it yields insights in the assumptions underlying the languages and how it can be used to produce a translation procedure between the languages.

1 Introduction

In this section we discuss the two main motivations for this paper: “Why do we need to compare languages?” and “Why do we need a new method for comparing languages?”

1.1 Why do we need to compare languages?

The field of KE has seen an increase of modelling languages in recent years [8, 2]. Such a large family of languages which all have roughly the same aim prompts the need to categorise and compare such languages, for the following reasons:

* This work has been (partially) funded by the Netherlands Computer Science Research Foundation with financial support from the Netherlands Organization for Scientific Research (NWO) within the REVISE-project, SION-project no. 612-322-316. The following members of the REVISE project contributed to this paper (listed in alphabetical order): Frances Brazier², Frank van Harmelen², Remco Straatman¹, Jan Treur², Niek Wijngaards², Mark Willems²

- To enable *choosing* the appropriate language for a particular application. Different properties of the languages make them suitable for different uses (e.g. for different types of applications, for different stages in the development process, etc). A comparison and categorisation helps to choose the appropriate language given a set of desired properties for a given purpose.
- To make it possible to *translate* a model from one language into another. Such translations are necessary to facilitate engineering in application projects, to increase re-usability of knowledge models between projects, and to enable use of different languages in different stages of a single project.

An important aspect in both these points is that each language embodies different assumptions about the objects and processes to be modelled, and about how the modelling primitives in a language should be used. These assumptions are often made implicitly, but both for choosing among and translating between languages it is essential that these different underlying assumptions are made explicit.

1.2 Why do we need a new method for comparing languages?

In the past few years, a number of attempts have been made to compare and classify KE modelling languages. These can be found in [3, 4, 5, 8, 2]. We will discuss the deficiencies of each of these attempts. Two parameters can be used to characterise all of these comparison attempts, namely the degree of similarity among the languages that were compared, and the amount of details that was studied in the comparison.

One of the first attempts at a broad comparison of KE modelling languages was the Sisyphus-I project [3, 4]. A number of languages was used to model the same simple task. The family of languages that was involved was very broad: Generic Tasks, KARL, OMOS, KADS, and others. As a result, the models that were constructed varied considerably, and only general properties of the languages could be studied. No final comparison paper appeared in the literature.

The VT-effort [5] also aimed at comparing a broad scope of modelling languages by modelling the same task in many languages. The problem of too little similarity among the set of languages was partly avoided by concentrating mainly on modelling domain and inference knowledge, and leaving control issues out of the comparison effort.

A third comparison effort was made at the ECAI'92 workshop on formal modelling languages for knowledge-based systems [8]. This effort again aimed at comparing a very broad set of languages ((ML)², MC, AIDE, KARL, DESIRE, OBJ3, MILORD, K_{BS}SF). Again, as a result, only rather general conclusions could be drawn.

Almost the opposite choice was made in [2]. This effort compared a family of closely related languages (OMOS, MODEL-K, MoMo, FORKADS, KARL, (ML)², QIL, K_{BS}SF, all aimed at KADS expertise models, and all being sufficiently precise for either formalisation or operationalisation). As a result, this study differed from the previous three in the greater amount of detail to which

the languages could be compared. The obvious limitation of this study was of course the restricted applicability.

All of this can be summarised as follows: often the underlying assumptions and aims of the languages were so different that they prevented a meaningful comparison, while in other cases the applicability of a meaningful comparison was limited to languages based on very strong and therefore limiting assumptions. In other words: current comparison efforts have been limited to either rather shallow comparisons on a broad-set of languages, or to detailed comparisons with a limited applicability to a narrow set of languages. The purpose of this paper is to propose a novel way of organising language comparisons. This new method for comparing languages will enable us to perform a comparison of languages at varying degrees of detail, allowing us a general comparison at high levels of abstraction, while not preventing more precise comparisons whenever possible. An explicit result of our comparison method will be insight in the different assumptions that underly the languages to be compared. Our method will enable us to detect when these assumptions correspond or differ, and, as a result, we can we can extend our comparison to the maximum level of detail permitted by the shared assumptions.

2 Comparison Method

The essence of the approach to language comparison that we advocate in this paper is that we begin by explicitly formulating the goals that each of the languages tries to achieve. Strange as it may seem, we do *not* take either the syntax or the semantics of the languages as a starting point for the comparison, but we concentrate instead on the *goals* that the language designers tried to achieve.

In itself, such a goal oriented approach to language comparison is not new: [8] already speaks about a “purpose-driven” comparison, and distinguishes formalisation and operationalisation as two distinct goals of separate sets of languages. The same distinction was made in [2] to divide the languages that are compared. However, after such a distinction based on language goals, both these papers subsequently concentrated on syntactic and semantic issues within each group.

The novelty of our approach lies in the fact that we use such language goals not only as a first division criterion, but that we use it as the sole basis for the entire comparison. In other words: our approach could justly be called a “purpose-driven” comparison method.

In somewhat more detail, our comparison method proceeds as follows: We first distinguish high level *goals* that the language designers have tried to achieve. In general, these will be quite abstract concepts such as “expressive power”, “executability”, “reusability”, “formal precision” etc. Notice that these are the types of distinctions also used in [8] and [2]. In our approach we then proceed to analyse which *methods* the language designers have employed to achieve these goals. These methods will be somewhat more concrete than the corresponding goals, and could be terms such as “compositionality”, “information hiding”, “separa-

tion of control”, etc. Each of these methods to achieve an abstract goal can itself be regarded as a somewhat more concrete goal, for which we can discern even more concrete methods, and so on recursively. At the lowest levels of this recursively alternating goal/method decomposition, we will arrive at very specific syntactic or semantic properties of the languages in question such as subprocedure constructs, global state representations, use of formal constructions, etc.

A crucial point is that such detailed language constructs appear in this goal/method decomposition tree *according to the reason why they were included in the language*, and not simply on the basis of a superficial or semantic similarity.

We will now discuss two important consequences that follow from such a goal-oriented comparison method, namely: - a measure for the degree of similarity between languages, and - a method for translating between languages

A measure for the degree of similarity between languages First of all, our comparison method gives us insight in the degree of similarity of the languages involved. When descending through the goal/method decomposition tree, languages that had corresponding high-level goals may well realise these goals in different ways, and therefore have different low-level goals. The level at which the languages begin to differ while descending through the goal/method tree is a measure for the similarity of the languages.

Notice that this measure of similarity cannot be derived by looking at the syntactic structure of the language: languages with very similar goals on quite concrete levels in the tree may well make different syntactic choices at the very lowest level, and therefore look superficially very different, even though they are in fact quite similar. Conversely, languages which have similar syntactic or semantic constructions may in fact be radically different when we look at how these constructions are meant to be used (ie. why they were included in the language).

A method for translating between languages A second consequence of our proposal for organising a language comparison is that it yields a translation method between the languages involved. This procedure works as follows: whenever we want to translate a construct c from language L_1 into language L_2 , we look up the construct in the goal/method tree for L_1 . We then traverse the tree upwards until we reach a node g which also occurs in the tree for L_2 . This means that apparently the construct c is used in both L_1 and L_2 for reason (goal, purpose) g . We then descend downwards from g into the tree for L_2 to find language constructs which are used in L_2 to realise goal g , and into which construct c should therefore be translated.

Notice that if construct c appears both in L_1 and L_2 , this does not mean that c from L_1 should also be translated as c in L_2 . After all, c may feature in L_1 for entirely different reasons than in L_2 . Instead, c from L_1 must be translated into a construct which is included in L_2 for the same reasons for which c was included in L_1 .

A further point to notice is that the effectiveness of this translation procedure is directly dependent on the degree of similarity between the two languages. If L_1 and L_2 are very different in their goals and in the methods they use to achieve these goals, then for many constructs c in L_1 , we will have to traverse up to quite high in the tree before hitting a node shared with the tree for L_2 . For such a high node (= an abstract goal), there will in general be very many constructions which contribute to realising that goal, which would render the method ineffective. This is of course exactly as expected, since translation mechanisms between two closely related languages will be more effective than those between distant languages.

Predictions From the above two consequences from our comparison method (a measure for language-similarity and a procedure for language-translation), we can derive two predictions that can be used to test the validity of our proposal.

1. If we take two related KE modelling languages, and we construct their goal/method decomposition trees, then these trees should coincide at the highest levels, and should begin to diverge at the lower levels in the hierarchy. Furthermore, the levels where the trees begin to differ should correspond to our intuitions on how similar the languages are and on the nature of their differences.
2. If we take a model expressed in one of the two languages, and apply the translation procedure described above, we should end up with a more or less natural description of the same model in the other language, or, alternatively, we should be able to explain precisely why certain constructs could not be translated in a natural way, in terms of missing correspondences between the goal/method trees for the two languages.

In the following sections, we will perform exactly these two experiments for two modelling languages for knowledge-based systems.

3 Goal graph

The design of (formal) specification languages and frameworks is, in general, based on a number of assumptions with respect to the goals the languages/frameworks pursue. A number of frameworks have been designed for modelling and specifying complex tasks in which reasoning plays an essential role, a number of which are mentioned above. The assumptions behind the design of two of these frameworks/languages, namely $ML^2/KADS$ and $DESIRE$ have been compared. The comparison has shown that the goals of the languages/frameworks are quite similar.

3.1 High level goals

At the top level five goals have been distinguished (the numbering refers to the graphs shown in figure 5).

- H1 transparency:** the structure of a system's architecture should be transparent: specifications should be comprehensible;
- H2 reusability:** specifications and architectures should be easily adapted to changing requirements, new knowledge, etc., to allow for reusability of specifications in different applications and/or tasks ;
- H3 expressive power:** a framework and specification language must be expressive: sufficient means must be available to model and specify the types of knowledge and interaction required;
- H4 establishing properties:** specification languages must allow for verification of system behaviour;
- H5 realizability:** specifications should provide sufficient detail to allow for implementation.

3.2 Lower level goals

Languages/frameworks may differ in the way in which the goals above can be pursued. To achieve these goals, lower level goals such as compositionality, play an important role. These lower level (sub)goals make the corresponding goals more concrete: they can be considered means to reach the higher level goals. Below each of the lower level goals distinguished will be described together with the relation to the more abstract goals with which they are directly associated. These goals are characteristic for the languages/frameworks considered and have been grouped together in a graph within which levels indicate levels of abstraction.

L0 Compositionality Modularisation is a generally accepted means to structure systems and specifications in both software and knowledge engineering: a means which is in line with the five main goals. Both ML² and DESIRE have been designed on the basis of this concept: complex reasoning systems are viewed as compositional systems within which complex tasks are modelled and specified as interacting subtasks and different types of knowledge are manipulated. This principle, compositionality, is essential to both frameworks. Together with the five goals characteristics, compositionality is one of the most important elements in the design of complex (reasoning) systems. Compositionality can be seen a central goal, and many of the other lower level goals are related to it. These other lower level goals are: (1) separation of knowledge types, (2) formal semantics, (3) partial specification, (4) operationalisation, (5) interactivity, (6) reflection and (7) partial reasoning, as described below. The relations among these lower level goals (as well as with the main goals described above) is also depicted in figure 5. The numbering refers to this figure.

L1 Separation of knowledge types Distinctions between the different types of knowledge involved, is one of the basic goals of compositional approaches to system design. The function of the knowledge involved is the basis for distinctions, and explicit separation of knowledge in system specifications.

The lower-level goals on which such distinctions are based are:

L1a case-specific/case-independent: Case-independent knowledge is used to reason about specific situations on the basis of additional information specifically related to the specific situations - case-specific information. For example, in a medical domain, information on a specific patient can be distinguished from general knowledge expressing relations between symptoms and diseases in the domain of medicine.

L1b separation of control Knowledge of control is explicitly distinguished from other knowledge involved. The interference of control knowledge with other knowledge may lead to systems that are not transparent and difficult to maintain. Separation of control knowledge, as part of the compositionality of a system can contribute to high level goals such as transparency and reusability. Also the high level goal of being able to establish properties of a specification is supported by separation of control knowledge.

L1c primitive components One of the principles on which compositional systems are based, is the principle that tasks can be decomposed and that a lowest level of decomposition can be assumed.

L1d layers Not only are different types of knowledge distinguished: different characteristics are often distinguished on the basis of which layers of knowledge are distinguished. The KADS three layer model distinguishes domain layer, inference layer and task layer. In DESIRE a knowledge dimension and a task dimension are distinguished.

L2 Formal semantics One of the guiding goals of both languages is that in order to establish properties of a system, it is necessary to formally express functionality, structure and behaviour.

L2a formal semantics of static aspects of reasoning For complex (reasoning) systems declarative, logic-based specifications are formulated with well-defined formal semantics. The goal of monotonicity for primitive subprocesses is a lower-level related goal.

L2b formal semantics of dynamic aspects of reasoning In many applications essential requirements are imposed on the dynamics of the system to be developed. During modelling, dynamic aspects are considered intensively. Therefore informal and formal semantics for dynamics are important. To this end temporal logic (for DESIRE) and dynamic logic (for ML²) are used. Lower level goals behind the formal semantics of dynamic behaviour include persistency, controlled inference and directed reasoning.

L3 Partial specification During modelling, partial specification is often useful: reuse of existing specifications is supported by existing partial specifications. Related lower-level goals are:

L3a information hiding Compositionality allows for information hiding, which in turn supports partial specification. Components can be developed independently from each other. Essential in this approach is that information in one component is hidden from other components. Information to be shared

with other components is specified in the interface of a component. A lower level goal is translation of elements of lexicons between components.

L3b generic (task) models Generic models are another means of using partial specifications. The specification is an abstract, domain-independent description of a task which can be specialised and instantiated.

L4 operationalisation One way to check whether the formal specifications of a system suffice is to implement a prototype on the basis of the specifications and to examine the behaviour. For some formal languages/frameworks operationalisation is a high-level goal, for which software environments support execution.

L5 interactivity Within complex interactive systems, interaction between systems and users should be explicitly modelled and specified. In the first place the lower-level goal of distinguishing agents is essential. For example, this can be fulfilled by defining separate components that model the different agents and that include knowledge of how to interact with each other.

L6 reflection In many problem solving processes explicit representation of reflective reasoning/meta-reasoning is of importance; e.g., to inspect and analyse in detail the current state of the process. To this end modelling techniques are required to be able to reflect on specific information and knowledge states of the system but also on task control.

L7 partial reasoning Reasoning is a dynamic process in which partial processes can be distinguished. To specify such reasoning compositionality is used (to separate or hide reasoning in one component from that in other components). The lower-level goal of controlled inference is of importance here.

L7a controlled inference In many complex situations exhaustive reasoning seldom occurs. Often reasoning processes proceed (a following task is begun) on the basis of a limited number of conclusions: either because a limited number suffice or because sufficient effort has been invested. Such factors are also explicitly modelled. The lower level goal of controlled inference is directed reasoning.

4 Using the comparison method to translate between languages

As mentioned in section 2, one of the expected results our comparison method would be the ability to perform translations between the compared languages. When translating between languages, it makes sense to translate parts of the source language playing a certain role (e.g. control knowledge) to the part of the target language that plays the same role. Our comparison method identifies which parts of the language are responsible for achieving a certain goal in the respective languages. Because of this we expect the comparison to be helpful in translating between languages.

The comparison-lattice describes, for each of the languages, which language elements are responsible for reaching a higher-level goal. If we want to translate a language element in the source language, we need to identify the language element(s) in the target language it must map onto. The appropriate element(s) can be found by tracing in the lattice which elements in the target language have the same parent goal as the source language element. It may both be the case that multiple elements map onto a single element and that multiple elements map onto a single element.

We will try to translate a specific, simple, DESIRE specification into a corresponding (ML)² specification. This experiment has two goals: (1) to evaluate whether the lattice helps in translating between languages, and (2) to try to gain insight in the relation between DESIRE and (ML)². Because of the second goal, we will also try to translate parts which are not described in the lattice (by using additional knowledge).

The next sections will describe the results of a case study in translating a DESIRE specification into a (ML)² specification. We start by describing the DESIRE model used for the translation experiment. Because of the two goals underlying the experiment, we will split our evaluation in two parts. First we will describe the approach followed and the problems experienced in translating DESIRE into (ML)². Next, we will discuss to what extent the comparison lattice was helpful in translating the specification.

4.1 The case: translating a DESIRE specification to (ML)²

The DESIRE specification used is that of a simple system for car diagnosis. We will only briefly describe the specification here ([6] contains the complete DESIRE specification and [7] the underlying motivation). The system uses *hypothetical reasoning* to come to a diagnosis. Hypothetical reasoning consists of determining a hypothesis, followed by the evaluation of this hypothesis. This continues until an acceptable hypothesis is found. Evaluating a hypothesis is done by determining which observations are consistent with the hypothesis, and comparing the actual observation with the expected observation. If a mismatch between these is found, the hypothesis can be rejected immediately. A hypothesis is confirmed when all observations match. The order in which hypotheses are tried, is predetermined. Which observations to test for a given hypothesis is determined by a causal network, that links hypothesis to observations.

4.2 Translating DESIRE specifications to (ML)²

This section describes the specific lessons learned with respect to the translation between DESIRE and (ML)². This section will be fairly detailed and some parts require knowledge of both languages.

Functional decomposition The DESIRE decomposition of complex components in subcomponents can be translated into the task decomposition of (ML)². Primitive components can be translated into primitive tasks and their associated primitive inferences in (ML)². As stated in the graph (see figure 5) the component

that models communication with the user (such as *obtain-observation-results*) can be modeled as a transfer task in (ML)².

Primitive components When translating DESIRE primitive components into (ML)², one has to consider different parts of the graph. First of all, the graph states that case independent knowledge is present in static roles in (ML)² and in the knowledge base of primitive components in DESIRE. Second, (ML)² also discerns domain dependent (domain layer) and domain independent knowledge (inference layer). The connection between these two layers is provided by static roles which redefine the domain knowledge in terms of the role it plays at the inference layer (for instance a disease at the domain layer is lifted to a hypothesis at the inference layer). So the knowledge-base of a DESIRE primitive component must be split into a domain-dependent part (which will be placed in a domain theory and is lifted to an inference layer static role) and a domain independent part (which will be placed in the inference body). An example of this translation is the knowledge base of *hypothesis-determination*. This knowledge base contains knowledge about preference between hypotheses, and knowledge on how to select the most preferred hypothesis that has not been selected yet.

```

knowledge base Hypothesis_Determination-kernel-kb
  better_hypothesis_than( broken_headlights, empty_battery );
  better_hypothesis_than( empty_battery, broken_starting_motor );
  if not rejected( broken_headlights )
    then selected( broken_headlights ) ;
  if not rejected( H: Hypothesis )
    and rejected( Hrej: Hypothesis )
    and better_hypothesis_than( Hrej: Hypothesis, H: Hypothesis )
    then selected( H: Hypothesis ) ;
end knowledge base Hypothesis_Determination-kernel-kb

```

Fig. 1. Hypothesis Determination knowledge base in DESIRE

It should be clear that the first type of knowledge ends up at the domain-layer in (ML)², and the second type in the inference body. Also note that since DESIRE does not make this distinction, human interpretation of the knowledge base is necessary to make this selection. Also additional vocabulary must be added for the static roles and/or the domain theory. Figure 2 depicts the translation.

A difference that is not stated in the graph is that in DESIRE primitive component can have multiple target sets. In (ML)² this would correspond to an inference which has more than one goal, which is not allowed. The DESIRE component with multiple targets (*hypothesis-evaluation* with targets *evaluate-to-confirm* and *evaluate-to-reject*) is modeled as two independent inferences in (ML)², one for each target. The data flow was adapted accordingly (since not all inputs were needed for both goals) which resulted in a less complex data flow.

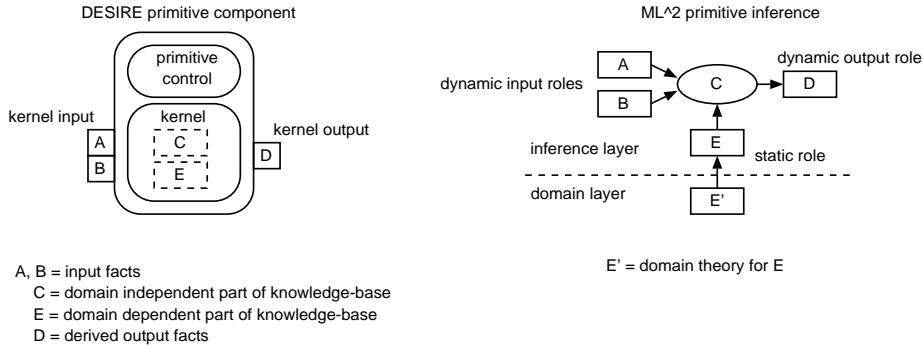


Fig. 2. Translating primitive components

Control over components As specified in the comparison graph, the task control knowledge-base in DESIRE and the task body in $(ML)^2$ are both responsible for the ordering of reasoning steps. In DESIRE control over the ordering of reasoning steps is specified by rules, whereas in $(ML)^2$ it is specified in procedural expressions based on QDL. However, since we know both representations are meant to provide the same function, we can easily translate between these representations. To this end, we first made control flow diagrams for the DESIRE task control database. Then, we constructed procedural programs that resulted in the same ordering of subtasks.

During task control translation, a difference between the languages emerged that was not described in the comparison graph. In DESIRE an explicit method for dealing with failing reasoning steps is present. In DESIRE, failure of a subcomponent to produce its target can be detected afterwards by the parent component by means of the `evaluation(<component>, <target set>, failed)` test, available in the task-control vocabulary. A complex component can also signal its failure by the `own-state(failed)`. In $(ML)^2$ this feature is missing, and one can only test *beforehand* whether a primitive inference will succeed for a given input (by means of the `has-solution_<pia-name>(<input>, <output>)` test). If a primitive inference is called in $(ML)^2$ and it fails, the entire system will fail. For a task no language elements are available to deal with failure of subtasks, or to denote its own failure.

Our solution to this problem was to adapt all tasks such that, in case of failure, they would return a special output value to denote failure. For primitive inferences we used the definition of primitive tasks (which in $(ML)^2$ are tasks that do nothing but call a certain inference) to mimic the behaviour of DESIRE primitive components. The body of the primitive task is defined such that it will test whether a primitive inference will fail (in which case the special output value is returned), and otherwise return the normal output of the primitive inference. Figure 3 shows a (part of) a $(ML)^2$ primitive task created this way.

```

ps-task-module hypothesis-determination
import
  hypothesis-determination ,
  definitions ;
input
  Eval-Hypos : evaluated-hypothesis set ;
output
  Sel-Hypo : hypothesis ;
signature
  programs
    hypothesis-determination : evaluated-hypothesis set ×hypothesis ;
task-structures
  ∀ Eval-Hypos : evaluated-hypothesis set
  ∀ Sel-Hypo : hypothesis
  (hypothesis-determination (Eval-Hypos , Sel-Hypo) ≡
    IF more-solutions-hypothesis-determination (Eval-Hypos , Sel-Hypo) THEN

      give-solution-hypothesis-determination (Eval-Hypos , Sel-Hypo )
    ELSE
      (Sel-Hypo := ∅ )
    FI );
end-ps-task-module

```

Fig. 3. Communicating failure of primitive inferences

Data flow between components When trying to translate the DESIRE data flow, the graph suggests that input and output roles in $(ML)^2$ have the same function as the input and output of components in DESIRE. Translation of information is done by translation of vocabulary and truth values by transformations in DESIRE, and by passing the info in roles in $(ML)^2$.

Data flow can be translated into $(ML)^2$ by “merging” inputs and outputs of components into intermediate roles. In fact, this difference between the languages stems from a difference in creation process. Aben [1] describes that during the construction of the inference layer *role ontology mappings* are necessary in order to be able to connect inferences which are selected from the library. These mappings ensure that the input roles of the consuming inference are compatible with the output roles of the producing inferences. So, whereas in DESIRE these mappings are part of the specification, in $(ML)^2$ they are part of the *process* of creating the specification. Figure 4 tries to clarify this distinction.

A further difference between the transformations in DESIRE and role ontology mappings in $(ML)^2$ is that DESIRE distinguishes object-meta(-meta-...,) relations between knowledge. In $(ML)^2$ this distinction is not present. In the translation, the object-meta distinction between knowledge is lost.

In the DESIRE specification a number of predicates were used to indicate where a particular input fact came from. Examples of this are the **selected** and **candidate** predicates for observations. In $(ML)^2$ these identifying predicates are not strictly necessary, since the source of inputs is identified by the input predicates in the primitive inferences. In order to stay close to the original DESIRE specification we included the identifying predicates, even though they perform no function in $(ML)^2$.

Finally, unlike DESIRE, the $(ML)^2$ data-flow does not strictly follow the decomposition. Communication between inferences is done strictly by means of

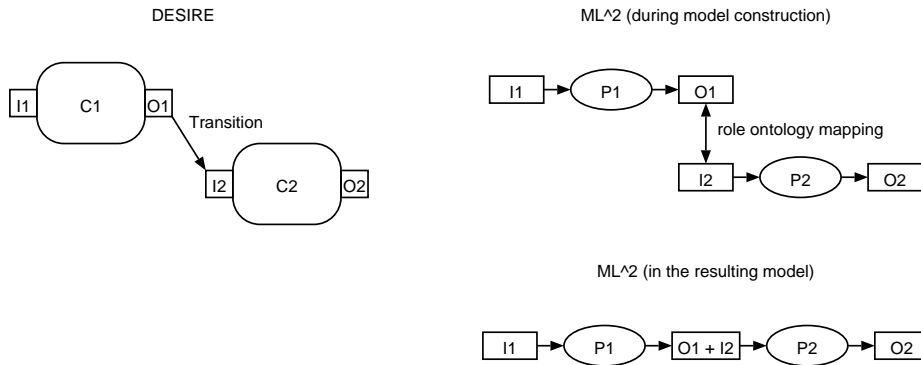


Fig. 4. Translating data flow between components

the roles at the inference layer, leaving an unclear role for the input and output variables of tasks.

4.3 Translating using the comparison graph

This section describes lessons learned regarding the use of the comparison graph in the translation process. It will use language specific examples only to illustrate what was learned.

In general, the graph was successful in identifying sources and targets for the translation tasks, and we were able to find suitable translations for the concepts we could identify counterparts in the other languages for. For instance, the comparison graph correctly identified that *DESIRE* task control knowledge bases should be translated into $(ML)^2$ task bodies. After this the actual translation between these elements was straightforward, despite the difference in representation.

On the negative side, we experienced a number of problems using the graph. These could be classified as: lack of guidance, errors in the graph, lack of detail, and concepts without counterpart. Of these, the first one is the most serious problem, the second and third problem are mainly problems of creating the comparison graph. The last one is a general problem for which no translation method can provide an answer.

- **Lack of guidance** The comparison graph does not give enough guidance in the ordering of the translation tasks. We first started out by picking some leaf of the graph and trying to translate that concept in *DESIRE* to the associated concept(s) in $(ML)^2$. After a few concepts were tried in this way, it became clear that translation could not be done in this arbitrary order. Because of the strong dependence of the various parts of an $(ML)^2$ specification certain translation tasks need to be performed before others. For instance, one cannot write down the task layer completely before the types

of roles at the inference layer are known. Since there is no way to derive an ordering from the comparison graph, we used additional information here. In [1] guidelines for translating informal KADS specifications into $(ML)^2$ are given. These guidelines also prescribe an ordering on this translation. In the DESIRE to $(ML)^2$ translation process we used this ordering, using the comparison graph to select appropriate DESIRE concepts for the $(ML)^2$ concept at hand. The ordering gave considerable guidance to the translation process and solved the above mentioned problem.

- **Errors in the comparison graph** Some errors in the original graph (that are now corrected) also confused the translation process. For example, the comparison graph indicates that both languages use “layers” as a way to distinguish between different types of knowledge. The graph then states that DESIRE uses object-meta distinctions for this purpose, and $(ML)^2$ uses a distinction between domain and inference layer for this purpose. Translation between these concepts in the respective languages is not possible though, since these are different distinctions between knowledge types. In fact, the comparison graph is not specific enough at this node and it should have contained sub-nodes. One sub-node for the distinction “domain-specific vs. domain independent” and one for the distinction “object-meta knowledge”. The same can be said for the nodes under “information hiding” which also deal with dissimilar concepts. In general, one should take care that children of a node really describe methods for the same goal.
- **Lack of detail** In general the graph in its current state lacks detail. A lot of language primitives are not present in the graph, which makes translation difficult. In our experience, the graph should contain all the language primitives to maximize its usefulness for translation.
- **Concepts without counterpart** Other problems in translating between the languages mainly consisted of concepts for which no counterpart exists in the other language. Clear examples for this are the above mentioned distinctions between “domain-specific vs. domain independent knowledge” (present in $(ML)^2$, but not in DESIRE) and “object-meta knowledge” (present in DESIRE, but not in $(ML)^2$). Translation between these concepts requires human creativity to, respectively, create and destroy the extra distinctions. However as stated in the introduction of this section this was to be expected.

5 Conclusions

For our conclusions, we will distinguish between the three different possible ways in which the combined goal-subgoal tree for two languages can be used, namely as a comparison method, as a distance measure, and as a translation method.

As a comparison method, our approach has distinct advantages over existing approaches. It allows a combination of both high-level and detailed comparison of the languages, whereas previous comparisons have always done either one or the other. Furthermore, our method encourages to abstract from syntactic

differences between the languages, and focuses on the goals and motivations that are embodied in the languages to explain the differences between the languages.

Concerning the use of our method as a difference measure, after our experiment we would not claim that our method provides a very precise notion of “difference” between two languages, but it is nevertheless useful as a general indication. The higher the level in the tree where two languages begin to differ, the further apart the two languages are. Furthermore, our difference measure differentiates between different aspects of a language: (ML)² and DESIRE differ early on in the tree concerning the operationalisation of the language, but are quite similar in other respects.

Regarding the use of our method as an aid in the translation process, we do not think that our goal-subgoal tree will ever lead to an automated translation process. Nevertheless, the general impression after the translation experiment was that the graph helped in decomposing the translation task by grouping related parts of the specification together. This led to clearly delimited subtasks in the translation process. On the negative side, we found that the graph gives no guidance with respect to the order in which the translation should be carried out (since no dependencies between the various fragments of a language are taken into account) and that the original graph lacked detail and was incorrect in places.

We are not in a position to compare the amount of work saved during our translation experiment with the amount of work it took to construct our graph. Further experiments would be required for such an analysis. Such future work should also indicate the applicability of this work to other languages, both inside and outside the field of knowledge modelling.

Acknowledgments

We thank the two anonymous referees for their helpful comments.

References

1. M. Aben. *Formal Methods in Knowledge Engineering*. PhD thesis, University of Amsterdam, Faculty of Psychology, February 1995. ISBN 90-5470-028-9.
2. D. Fensel and F. van Harmelen. A comparison of languages which operationalise and formalise KADS models of expertise. *The Knowledge Engineering Review*, 9:105–146, 1994.
3. M. Linster. Sisyphus’91 part 2: Models of problem-solving. statement of the sample problem. In D. Smeed, M. Linster, J. H. Boose, and B. R. Gaines, editors, *Proceedings of EKAW91*, Glasgow, 1991. University of Strathclyde.
4. M. Linster. Sisyphus’91/92: Models of problem solving. *Int. J. of Human Computer Studies*, 40(3), 1994. Editorial special issue.
5. A. Th. Schreiber and W. P. Birmingham. The Sisyphus-VT initiative. *International Journal of Human-Computer Studies*, 1996. Editorial special issue.
6. R. Straatman, F. Brazier, F. van Harmelen, J. Treur, N. Wijngaards, and M. Willems. A purpose driven method for language comparison. Revise project, University of Amsterdam and Free University of Amsterdam, 1995.

Fig. 5. The goals and subgoals of $(ML)^2$ and DESIRE

This article was processed using the \LaTeX macro package with LLNCS style