Temporal Abstraction Mechanisms as KADS Inferences

Manfred Aben[†]
Yuval Shahar[‡]
Mark A. Musen[‡]

[‡]Department of Social Science Informatics
University of Amsterdam
Roetersstraat 15, 1018 WB Amsterdam, The Netherlands
e-mail: manfred@swi.psy.uva.nl

[‡]Medical Computer Science Group
Knowledge Systems Laboratory
Stanford University School of Medicine
Stanford, CA 94305–5479, USA
e-mail: shahar@camis.stanford.edu

1 Introduction

In [Aben, 1993b], one of the present authors discussed various problems with primitive inference steps in KBS design. One of the issues addressed in that paper is the fact that the chosen domain–ontology determines the reusability and usability of the primitive inference steps (from now on called *inferences*).

One of the solutions proposed there is to define multiple ontologies, each suited for a particular task or range of tasks, and to map these ontologies onto a generic, low–level ontology that can be used to describe arbitrary domain knowledge. This base–ontology would then assure reusability of the inferences that can be defined in terms of this ontology, while the task–specific ontologies would allow for a more usable definition of the inferences.

In this paper we will substantiate this claim by discussing a mapping of the temporal abstraction ontology and temporal abstraction mechanisms as described in [Shahar *et al.*, 1992] onto the KADS base–ontology and KADS primitive inferences as described in [Aben, 1993a].

The structure of this paper is as follows. Section 2 discusses the need for both base–ontologies and task–oriented ontologies, and a mapping between these ontologies. Section 3 describes the notion of KADS inferences in terms of operations on ontologies. Section 4 describes the PROTÉGÉ–II temporal abstraction ontology and mechanisms and maps them onto the KADS base–ontology and inferences respectively. Section 5 lists the results of this exercise.

2 Ontologies

In order to describe reasoning patterns and expertise, we need to establish a language and terminology that is both sufficiently expressive and concise. A third, and underlying

requirement is that we also want to write down the expertise in a terminology that is close to the task or expertise at hand.

Ontologies can be distinguished in terms of their breadth and depth. *Deep* ontologies cover many aspects of the task at hand, whereas *broad* ontologies cover a wide range of different tasks. Deep ontologies are therefore very *usable* for a particular task, but broad ontologies are highly *re–usable* across many tasks. Deep ontologies which are tuned to a particular task may be very efficient and effective to describe that task, but typically will not be of much use outside the scope of the task(s) they where designed for. On the other hand, broad ontologies consist of very small–grained categories, in order to make them useful across a wide range of tasks. This makes the use of such an ontology very cumbersome, and the resulting expressions hard to understand.

As both *usability* and *re–usability* are acknowledged to be of vital importance to of KBS design, we will have to find a way to reconcile the two. This reconciliation of ontologies of various grain–sizes and various scopes of applicability can be achieved by *ontology mapping*. Ontology mapping is the expression of one ontology in terms of another. In this case we are particularly interested in mapping special purpose ontologies onto general purpose ontologies. Special purpose ontologies are a convenient shorthand for more complex expressions in the general purpose ontology.

In the next sections we present the KADS base ontology as an example of a general purpose ontology, discuss special purpose ontologies, and a mapping between the special purpose ontologies and the general purpose ontology.

The KADS base ontology

The KADS base–ontology is an example of a broad, general purpose ontology to describe domain knowledge[1]. This ontology was proposed in [Breuker *et al.*, 1987a], extended in [Wielinga *et al.*, 1992] and formalized in [Aben, 1993a]. It is inspired by the epistemological primitives proposed by [Brachman & Schmolze, 1985]. The base–ontology consists of the following elements.

attributes are descriptive primitives, they are the `color` building blocks of descriptions of concepts that can be distinguished in the domain.

Attributes are formalized as functions over instances: f : instance $\longrightarrow$ value.

values can be assigned to attributes. `red` Values are primitive, i.e. they have no internal structure.

A value v is the result of a function application on an instance: $f(\iota)$.

expressions are defined over attributes and values and the concept or instance to which they are defined. The most typical value expression is the application of the attribute to a concept. Example `:color(bicycle)`.

An expression is formalized as an n–ary *function* over the values of attributes. The result of a value expression is itself a value, and hence may be used as an argument to another value expression. Expressions are denoted as: $\epsilon(f_1(\iota), \ldots, f_n(\iota))$.

[1] Another example is the ontology of *Situation Specifc Models* as proposed in [Clancey, 1992].

value–relations are relations between attribute expressions including *attribute–value pairs* and *value restrictions.* In practice, *expressions* will only occur in value–relations. Value–relations have a truth–value. Example: `color(bicycle)` $\in$ {`red, green, blue`}.

Value relations are formalized by an n–ary *predicate* defined over value expressions. A typical example of a value relation is a *value restriction*. Value relations are denoted as: $\varrho(\epsilon_1, \ldots, \epsilon_n)$. Value relations can also be higher order, *i.e.* they may be defined over other value relations: $\varrho(\varrho_1, \ldots, \varrho_n)$.

Attribute value pairs are specific value relations, the relation in this case is equality. Hence they are denoted as equations of function applications and terms: $f(\iota) = v$. To specify that some attribute is defined on some instance, without knowing, or caring about, the value of that attribute, we write: $f(\iota) \downarrow$, which is a shorthand for: $\exists\ V\ f(\iota) = V$. To specify that some attribute is not defined on some instance we write $f(\iota) \uparrow$, a shorthand for $\neg\exists\ V\ f(\iota) = V$. Note that attributes are defined on instances only. In order to specify that an attribute is defined over all instances of some concept, we can write $\forall I\ c(I) \rightarrow f(I) \downarrow$, and we introduce a shorthand notation $f(c) \downarrow$. Similarly we define $f(c) \uparrow$ and $f(c) = v$.

concepts Concepts are defined by their name and their attributes (possibly with value–restrictions); concepts are essentially *frames*.
Example: `bicycle: [nr_of_wheels = 2,color,size,type]`

A concept is formalized as a unary predicate of which the argument is an instance, or a variable that ranges over a set of instances. Concepts are denoted as $c(I)$, or, when the instances on which the concept is defined are not relevant, abbreviated to c.

instances Instances are *subsumed* by their concepts, i.e. all attributes defined on a concept are defined on its instances, and have the same or a more specific value. Instances are defined by their name, the concept to which they belong and their attribute–value pairs. The internal structure of concepts and instances are determined only by the attribute–value relations.
Example: `my_bike:bicycle[nr_of_wheels = 2,color = red,size = 14",type = ATB]`

An instance is formalized as a constant ι.

relations Relations are defined between concepts or between instances, such as the *is–a* and *part–of* relations. Example: `part-of(bicycle, front-wheel)`

A relation is denoted as an n–ary predicate defined over instances, where $n \geq 1$: $r(\iota_1, \ldots, \iota_n)$ is called a *relation instance*. To indicate that all instances of some concept are related to all instances of another concept, we may write: $\forall I_1, I_2\ c_1(I_1) \wedge c_2(I_2) \rightarrow r(I_1, I_2)$ or as a shorthand $r(c_1, c_2)$.

structures Structures consist of concepts or instances, and their part–of relations, *i.e.* they can be seen as sets of relation instances. Attributes can be defined on whole structures as well. Sets, as well as lists, are specific instances of structures. Example:

```
bicycle:[covers-all-bikes = no]
{part-of(bicycle, front-wheel),part-of(bicycle, back-wheel)}.
```

A structure $\mathcal{S}$ is defined and denoted by a set of relation instances:
$\mathcal{S} = \{r_1(\iota_1, \ldots, \iota_n), \ldots, r_k(\iota'_1, \ldots, \iota'_m)\}$.

A list L is an ordered set of instances without duplicates, embraced by angle brackets: $\langle \iota_1, \ldots, \iota_n \rangle$. Since the list is ordered, there is a relation between all successive instances in the list. Hence, a list is a specific type of structure, where all relation instances are binary relations over successive instances in the list: $\langle \iota_1, \ldots, \iota_n \rangle \equiv \{r(\iota_1, \iota_2), \ldots, r(\iota_{n-1}, \iota_n)\}$ for some ordering r.

Task–oriented Ontologies

The KADS base–ontology is domain– and task independent. This is an advantage because it gives a small set of primitives which seem adequate to model many problem types and areas. However, the base–ontology is not specifically suited to model the *roles* that the domain knowledge plays during problem solving. As we will see below, inferences are defined as operations on ontological characterizations of the domain knowledge. Hence, if the chosen ontology is very task–independent, the resulting inferences can only be formulated in cumbersome ways.

On the other hand, the *roles* that domain knowledge elements play during problem–solving are merely labels, without an explicit semantics. If we can fix a set of such labels, and establish which primitives of the KADS base–ontology constitute the knowledge that the labels refer to, we can then formalize inferences over these roles, in terms of the KADS base–ontology. This relies on the assumption that knowledge roles can indeed be expressed in terms of the KADS ontology.

In [Benjamins, 1993] a task–specific role–ontology for diagnosis is presented. This ontology consists of terms like *hypothesis, observation, contributor, finding etc.* Benjamins assumes that the suitability criteria that determine the applicability of a problem solving method can be mapped eventually onto domain specific terms, but no explicit mapping onto an abstract domain ontology is given.

Ontology mapping

The various approaches to KBS design use different ontologies to describe the knowledge that is used by their building blocks. The ontologies range from very task–specific (Generic Tasks [Chandrasekaran, 1986], Component of Expertise [Steels, 1990]), to very use–independent like the KADS base–ontology. There is a trade–off between the descriptiveness of the ontology, *i.e.* wether it coincides with the terminology of the task that is modeled, and the reusability of the ontology, *i.e.* wether it can be used to model various tasks. The reusability of the building blocks is related to their grain-size, which is in turn related to the differentiations that can be made by the ontology.

One can imagine layers of ontologies, where each layer is formulated in terms of a lower-level, more widely applicable ontology. In [Schreiber *et al.*, 1993; van Heijst *et al.*, 1993] an approach to medical KBS construction is described which distinguishes between three ontologies, and mappings between them:

- The **Application Knowledge** specifies the actual domain knowledge terms and relations between them (called *domain vocabulary* and *domain models* respectively).
- The **Application Ontology** specifies the structure of the domain in terms of domain knowledge *types*.
- The **Knowledge Roles** specify the names for domain knowledge elements that characterize their role during problem solving.

The *base ontology* would provide the abstract terminology (or *frame–ontology* in the terminology of Ontolingua [Gruber, 1993]) for the *application knowledge.* The *abstract, intermediate* ontology coincides with the *application ontology* and the *task–oriented ontology* coincides with the *knowledge roles.*

Most approaches distinguish between a base–ontology and a task–specific ontology that can be mapped onto the base–ontology. Hence a set of generic inferences can be specified in terms of the base–ontology. The role–limiting method approach, the components of expertise approach, and the generic task approach, all predefine the mapping between the inferences that operate on the base–ontology, and those that operate on the task specific ontologies. The PROTÉGÉ–II system, and to a larger extent, the KADS methodology consider it part of the modeling process to construct this mapping. These mappings are never one–to–one, and the support that the various methodologies provide to make these mappings is limited.

The Knowledge Sharing Effort [Neches *et al.*, 1991; Patil *et al.*, 1992] is a large scale project with the aim to make the enormous amount of knowledge in many of todays knowledge based systems available to the AI community as a whole. In this context, the *OntoLingua* [Gruber, 1993] system has been developed to support the expression of ontologies in the *Knowledge Interchange Format (KIF)* [Genesereth & Fikes, 1992]. *OntoLingua* provides a basis to formalize ontological commitments made in various systems and tasks. The work described here shares the goal of knowledge reuse, and in particular problem solving method re–use. The difference between the *OntoLingua* approach, and the ontology mapping approach discussed here is mainly in its goal. *OntoLingua* is meant to translate to and from specific knowledge representation languages and systems, whereas we translate between various ontologies. From a syntactical viewpoint this difference may seem minute, but it does have semantical implications. We explicitely do not suggest any particular conceptualization, but we argue that various conceptualizations are possible for any particular task or domain, and that the relation between these conceptualizations should be explicitely defined. It would be a trivial task, however to formulate the ontologies presented here in *KIF*.

3 Inferences

In KADS ([Schreiber, 1992; Wielinga *et al.*, 1992]) KBS design is seen as a modeling process. The central model resulting from this process is the expertise model. The *expertise* model consists of three layers, each having their own structuring primitives.

The *domain layer* describes all static, axiomatic domain knowledge. One of its structuring primitives is the *domain ontology*, which gives the terminology in which the domain knowledge is expressed.

The *inference layer* describes the possible inferences that can be drawn on the basis of the domain knowledge. The primitives at the inference layer are the (primitive) *inferences* and *knowledge roles*. Inferences operate on domain knowledge through the knowledge roles, which are labels describing the role that a particular domain expression plays during inferencing. Since inferences and roles are of central focus in this paper, we will give some definitions and dogmas below.

The *task-layer* describes control over the inference steps at the inference layer. The primitives here are *task–decomposition*, which allows for specifying control over groups of inferences, and *control structures*, which are the control structures known from imperative programming languages including non-determinacy.

Some definitions

Definition 3.1 (Knowledge Role) *A* Knowledge Role *is a label that indicates the role that domain knowledge to which the label is attached, plays in an inference process.*

Definition 3.2 (Inference) *An inference is the declarative definition of the relation between* input knowledge roles *and* output knowledge roles *in an inference process, possibly by means of* support knowledge roles*. The only references that an inference makes to the domain knowledge are through the input, output and support knowledge roles. The only domain knowledge that is affected by the inference is the domain knowledge that is referred to by the output knowledge role.*

Given this definition, the following dogmas should be respected when defining a particular inference.

Dogma 3.1 (Ontological requirements) *Knowledge roles have no internal structure, they are defined solely by their names. However, in order to describe the inferences other than just describing abstract relations between* labels*, the roles include the ontological requirements on the domain knowledge that plays the particular role.*

Dogma 3.2 (Internal control) *As inferences are* declarative *descriptions of I/O relations, they should not specify any control, indicating how the inference is applied.*

Dogma 3.3 (Domain independence) *As stated in the definition, the* only *references that an inference makes to domain knowledge should be made through the knowledge roles.*

Dogma 3.4 (Level of detail) *Inferences should describe all, and only,* interesting *inferencing. What amounts to "interesting" is a modeling decision, and depends on a particular modeling situation. Typical "trivial" inferencing includes using transitivity of some relation in order to establish whether a relation holds, etc. This type of knowledge should be defined at the domain–layer.*

Inferences as operations on ontologies

Inferences operate (indirectly) on domain knowledge. The conceptualization of domain knowledge consists of two steps: the establishment of the universe of discourse, i.e. the

domain ontology, and the establishment of axioms over elements of this ontology. Inferences are defined as operations on ontological primitives, by which the domain knowledge is conceptualized, and they may *refer to* or *use* the domain axioms. Hence, in order to describe the inferences, we first have to describe the abstract domain ontologies. The ontologies presented above should be seen as examples, as other ontologies are conceivable. The main point is that different views on domain knowledge may require different ontologies, and that the relations between these ontologies should be clear.

KADS provides a set of inferences (cf. [Breuker *et al.*, 1987a]) which are specified in terms of the base–ontology. These inferences model operations on ontological categories of domain knowledge, but they do not model operations on the "roles" that domain knowledge plays in the inference process. On the one hand, this is an advantage because it gives a small set of primitives which are adequate to model many problem types and areas, hence supporting knowledge reusability. On the other hand, the generality of the ontology makes it often difficult to model an inference. This is because one specifies the inferences as operations on the *roles* that the domain knowledge plays in the inference process, and not in terms of their *ontological structure*.

Because the base–ontology does not describe the *roles* that ontological categories play in the inference process, the inferences described in such a way are of a rather syntactical nature; they describe how ontological categories are manipulated, but not how specific knowledge roles are manipulated. On the other hand some of the inferences that are used to model specific tasks *are* task and/or domain related, but others *are not*. For instance, the rather syntactic inference **select**, which selects an element from a set, is often used alongside more task–oriented inferences.

The choice of an ontology also has consequences for the set of possible inferences. Inferences specify operations on the underlying ontological categories. Hence each abstract domain ontology implies a set of possible inferences, which can be classified with respect to the ontological categories they manipulate. This was the basis of the original classification of KADS knowledge sources (cf. [Breuker *et al.*, 1987a]). As we have seen in the above, we want to abstract from the KADS base ontology and define other ontologies. Each of these ontologies still has to be reusable, and mappable on the base ontology. Therefore each particular abstract domain ontology implies a particular set of possible inferences.

How are inferences described?

We have defined the inferences as defining the relation between input and output roles. In order to clearly define inferences, we can distinguish various *constituents* of an inference. These constituents separate the parts of the inference definition that play a distinct role.

Preconditions Preconditions describe the conditions under which an inference is applicable. We can use preconditions to separate detailed restrictions on the input– and support knowledge roles from the actual operations performed by the inference.

Inference Body The operation performed by the inference is described in the inference body. The inference body specifies the relation between the inference's input and output roles, with the use of the support roles.

Postconditions Postconditions describe the properties of the output knowledge roles, or other conditions that should hold after application of the inference, such as a reduction of the search space.

Intuitively, the "defining" knowledge of an operation should be in the body, and the "consequential" knowledge should be in the postcondition. This would imply that the postcondition is logically redundant with respect to the body. However, the postcondition should provide a "window" on all possible consequences of the body: it should consist only of these consequences that might be useful to select the inferences. Note that we might decide to specify a set of postconditions, each representing a separate index to the inference.

In [Aben, 1993a] a formal treatment of the various constituents of an inference is presented.

Relation Between Inferences: Inference Schemata

Some inferences are more specific than others, in terms of what they *do*. For instance, selection of an instance from a set can be specialized by requiring that the selected instance has some unique property. Using this type of dependencies between inferences can serve as an aid for selecting and gradually refining the appropriate inference for a given task. Since all inferences described in [Breuker *et al.*, 1987a] can be differentiated further, it is better to regard these inferences as inference *schemata* or *types*. Knowledge engineers using the KADS methodology should be provided with a library of modeling components. On the basis of [Aben, 1993a] we argue that such a library will not contain "ready–to–use" components, but rather a set of adaptable schemata.

If we describe inferences as a combination of their constituents as mentioned above, we can establish how they relate to one another. One inference may be for instance more general or more specific than another, or its precondition may be stronger than that of another inference. In [Aben, 1993b] a more elaborate overview and formalization is presented of these inter–inference relations.

4 Mapping PROTÉGÉ–II mechanisms onto KADS inferences

In the PROTÉGÉ–II framework, the basic components for the design of knowledge based systems are the *methods* and *mechanisms*. Methods can be decomposed into *methods* or bottom out in *mechanisms*. This decomposition is similar to the task–decomposition in KADS. In KADS, the *inference structures* form the counterpart of PROTÉGÉ–II's problem solving methods, and the basic building blocks in a KADS inference structure are the *inferences*.

The major difference between KADS inference and PROTÉGÉ–II mechanisms is in their grainsize. In KADS the decomposition bottoms out in inferences, which are *primitive* in the sense that no internal control is specified. The control–aspects of a problem solving method are described in a separate control structure at the task layer, which enforces control over an inference structure. The PROTÉGÉ–II mechanisms *do* specify (internal) control, hence one would therefor expect some difference in grainsize between mechanisms and inferences.

Although much can be said about the mapping of the control issues and static domain knowledge between both frameworks, in this paper we focus on mapping the mechanisms and inferences only. However, as inferences as well as mechanisms operate on (ontological characterizations) of domain knowledge, we will first have to outline the mapping between the ontologies used by the specific mechanisms for temporal abstraction onto the KADS base–ontology.

The next sections will describe the temporal abstraction ontology and mechanisms as presented in [Shahar *et al.*, 1992; Shahar *et al.*, 1993], and map these descriptions onto the KADS base–ontology and inferences. Since the mechanisms are of a larger grainsize than the KADS inferences, the mappings map onto KADS inference structures.

4.1 The temporal abstraction ontology

In [Shahar *et al.*, 1992], and in more detail in [Shahar & Musen, 1993] a *task–oriented ontology* for temporal abstraction is presented, which consists of the following terms:

- A *time–stamp* τ may be an integer, or a more complex structure.
- A *time–interval* $I_i \equiv [\tau_s, \tau_e]$ is determined by two time-stamps, with $\tau_s \leq \tau_e$.
- A time–point T_i is a zero-length time-interval where $\tau_s \equiv \tau_e$.
- A *parameter* π is an attribute–value pair associated with some object (e.g., a patient). Parameters have types that denote the scale on which they are measured: nominal, ordinal, interval, or ratio.
- A constant parameter is an atemporal parameter (e.g., gender).
- *primitive parameters* are "attached" to *time-points* only. They can be seen as raw sensory data (e.g., hemoglobin counts).
- *abstract parameters* can be attached to *time-points* and to *time-intervals*. They can be seen as *characterizations* of the intervals to which they are attached (e.g., `low` hemoglobin). As we will see, these characterizations have different semantics than Allen's properties [Allen, 1984][2].
- A *state* σ is the conjunction of a time-point or a time-interval and a parameter. We will associate two temporal attributes to a state, the *start–time* and the *end–time*, which together represent the time–interval (or time–point if both attributes are equal) in which the parameter has a particular value.[3]
- An *abstract state* is the conjunction of a time–point or time–interval and an abstract parameter.
- An *intervention* is an external action that can be controlled by the agent (e.g., administration of a particular drug).
- An *event* is a conjunction of a time–point or a time–interval and an intervention.

[2]The distinction between abstract and primitive parameters is a convenient, but not crucial, one. It signifies the fact that input data are usually primitive parameters, whereas conclusions that are derived by the problem solving method from values of other parameters are abstract. In some contexts, however, and abstract parameter might be given sometimes, or even always, as part of the input, and thus would play the roles of a primitive parameter for that particular task.

[3]We will use the terms *point–state* to denote that the parameter characterizes some time–point, and *interval–state* to denote that the abstract parameter characterizes a time–interval.

4.2 The temporal abstraction mechanisms

[Shahar *et al.*, 1992] defines three types of temporal abstraction: *point temporal abstraction, temporal inference* and *temporal interpolation*. Below we will reiterate their definitions, and we will also present them as KADS inference structures. For reasons of space, we cannot describe in detail all the inferences that occur in the inference structures, but the reader is referred to [Breuker *et al.*, 1987b] for an informal description, and to [Aben, 1993b] for a formalisation thereof.

In temporal abstraction, we actually abstract the *parameter* rather than the time–interval itself, although we may generalize the temporal aspects of the state into, for instance, larger intervals on which we derive the abstracted parameter to hold.

point temporal abstraction

Input: point–state (set)

Output: abstracted point–state

Description: The point–states that are input to this inference are assumed to be *temporally equivalent* under an equivalence relation ρ (e.g., the same hour every day of the week). The primitive or abstract parameter(s) on time points T_i are mapped onto an *abstract* parameter on the time point to which all time–points are equivalent. *Point temporal abstraction* abstracts the *parameter(s)*, and maps all equivalent time–points of the point–states onto one time–point.

There are two types of *point temporal abstraction*: *classification p.t.a.* which classifies a parameter into an abstract parameter, and *computational transformation p.t.a.* which abstracts one or more (primitive or abstract) parameters onto a single abstract parameter and involves additional computation.

temporal inference

Input: one or two (abstract) interval–states; possibly a time-interval

Output: (abstract) interval–state

Description: Temporal inference takes one interval–state and one time-subinterval (to which the abstract parameter might be attached), or two meeting interval-states (which might be concatenated) and produces a sub– or superinterval of the input interval–state(s). The input interval–states are assumed ordered, i.e. given n intervals $[\tau_{s_1}, \tau_{e_1}], \ldots, [\tau_{s_n}, \tau_{e_n}]$ output one interval $[\tau_s, \tau_e]$ where: $\forall_{1 \leq i \leq n}\ \tau_{s_i} < \tau_{e_i}$ and $\forall_{1 \leq i \leq n}\ \tau_{s_i} \leq \tau_{s_{i+1}}$

Abstract parameters can have three semantic *inferential properties*, corresponding to three *inference types* that are defined. The inferential properties are a modification of the temporal semantic properties of propositions as suggested by Shoham [Shoham, 1987]. The inference types are:

- *downward hereditary* (one input interval-state): $\tau_s \leq \tau'_s \wedge \tau'_e \leq \tau_e$
  (The characterization of the interval I holds over every subinterval of I.)
- *concatenable* (two input interval-states): $\tau'_s = \tau_{s_1} \wedge \tau'_e = \tau_{e_2}$
  (The characterization holds over the union of two meeting intervals I_1, I_2.)
- *upward hereditary* (one input interval-state): $\tau'_s \leq \tau_s \wedge \tau'_e \geq \tau_e$
  (The characterization holds over every interval I_2 that includes I_1. Note that this is different from the use of this term by Shoham.)

An *inference properties table* can be defined as a six–tuple

$$\langle \pi, \theta, v, \phi, \tau, \xi \rangle$$

where:

- π is the *parameter* to be abstracted,
- θ is the *abstraction type*, one of *state, gradient* or *rate abstraction*. State abstraction classifies the magnitude of parameter (i.e. parameter abstraction). Gradient abstraction classifies the sign of the derivative of an (abstract) parameter during an interval into a nominal value (typically one of {`same, increasing, decreasing, nondecreasing, monotonic, nonmonotonic`}. Rate abstraction classifies the magnitude of the derivative of an (abstract) parameter into a nominal value (typically one of {`slow, fast, stable`}. Gradient and rate abstraction are only meaningful if π has an ordinal type. Note that the θ abstraction of a parameter is also a parameter.
- v is the *value* of π,
- ϕ is the *inferential property*, i.e. one of *downward hereditary, upward hereditary* or *concatenable*,
- τ is the *truth–value* of the inferential property ϕ, given the other elements of the six–tuple, and
- ξ is a *context*, formed by events or states which includes external events, history or other information.

A *join* operator $\otimes$ can be defined that joins intervals, creating a new interval, attached to which is a new parameter value that is a computational transformation of the parameter values attached to the two input intervals. For example, in the case of the gradient abstraction, we could define `decreasing` $\otimes$ `increasing` = `nonmonotonic`. We call the table that describes all such combinations an *abstraction-inference table*. The abstraction type of both joined intervals should be the same (e.g., state).

In [Shahar *et al.*, 1992] a default abstraction-inference table for the gradient abstraction is defined that includes the gradient values `same, increasing, decreasing, nondecreasing, nonincreasing, nonmonotonic` such that the $\otimes$ operator is *associative* and *commutative*. For instance, `increasing` $\otimes$ `same` = `nondecreasing`. The associativity is an important feature since we do not assume any particular order of application of the temporal abstraction mechanisms.

The abstraction–inference table contains tuples:

$$\langle \pi, \theta, v_1, v_2, v_3, \xi \rangle$$

where v_1 is the value of π on the first interval, v_2 on the second interval, and v_3 on the joined interval. This tuple is defined only when the concatenable inferential property is true for this parameter with the value v_3 in that context; i.e., the tuple $\langle \pi, \theta, v_3, \texttt{concatenable}, \texttt{true}, \xi \rangle$ exists in the inference-properties table.

In the particular case of using temporal inference for concatenating states, we assume that $[\tau_{e_1} \equiv \tau_{s_2}]$ (meeting intervals); this constraint is generalized by the temporal-interpolation mechanism.

temporal interpolation

Input: two (possibly abstract) states

Output: abstract interval–state

Description: Temporal interpolation involves joining states, into an interval–state and abstracting the parameter(s). Temporal interpolation requires that the input–states are *concatenable* and the values of the input–parameters can be *joined.* There are six types of temporal interpolation: For each θ *(state, gradient* or *rate)* one can define a *primary* θ interpolation and a *secondary* θ interpolation. Primary interpolation joins two point–states into a interval–state, whereas secondary interpolation joins two interval–states into one interval–state.

The maximum gap allowed between intervals I_1 and I_2 is a context–, domain– and task–dependent function:

$$\Delta(\pi, \theta, v, |I_1|, |I_2|, \xi)$$

where v is the abstracted value of π on $I_1 \otimes I_2$. If the gap between the input states is less than this Δ–function, then I_1 *extends* I_2 ($I_1 \approx I_2$). The maximal-gap Δ function represents domain- and task-dependent knowledge about the rate of change and, therefore, about the persistence of truth, of the characterization v_3. McDermott's persistence assumption [McDermott, 1982] and McCarthy's inertia principle [McCarthy, 1986] can be seen as private cases where $\Delta(\ldots) = \infty$.

The *abstraction-interpolation table* is an extension of the abstraction-inference table, defining the interpolation operation for every abstract parameter and combination of abstraction values. This table describes interpolation conditions additional to the maximum time-gap, such as when a variation in the value of π is considered an increase. Every parameter π is assumed to have a domain-dependent, context-sensitive, "significant-change" value C_π that denotes the maximal increment below which a change is not considered an increase or a decrease. C_π may actually be a function of π, the initial value of π and the context ξ.

Temporal interpolation is an extension to temporal inference for the special case of bridging a time-gap, where both point–states or interval–states have the same

abstraction–type, and its conclusions are somewhat stronger than those of temporal inference. For instance, the gap between two `increasing` interval–states might be bridged to form a joined `increasing` interval–state, even though there was no increase (but no significant decrease) in the value of the abstracted parameter π during the gap interval.

The case of forming gradient or rate abstractions from abstract point-states that were formed by a multi-parameter point temporal abstraction is a special one and is discussed elsewhere [Shahar *et al.*, 1992].

4.3 Mapping the TA ontology onto the KADS base-ontology

The temporal abstraction ontology can be mapped onto the KADS base–ontology as shown in table 1, which also introduces the symbols that will be used in the rest of this paper to denote the various ontological categories.

symbol	TA category	KADS category
σ	state	**concept**
Σ	state set	**set** of **concept**s
π	parameter	**attribute** of *state*, type *value*
Π	parameter set	**set** of **attribute**s
ρ	relation	**relation**
ξ	context	**set** of (**relation**) **instance**s
τ	time	**concept**
τ_s	time–start	**attribute** of *state*, type *time*
τ_e	time–end	**attribute** of *state*, type *time*
θ	inference–type	**concept**

Table 1: Mapping the temporal–abstraction ontology onto the KADS base–ontology

Obviously, many other mappings are possible. One could model parameters as **concept**s, and a specific parameter/value instance as a **relation instance**. The preferred mapping is usually the one that shows as much structure of the objects of interest as is required, but no more. Remember that the most primitive non–decomposable elements of the base–ontology are the **attributes** and **values**, followed in level of structural complexity by **instance**s, **concept**s, **relation**s and **structure**s.

It is clear from the table that the mapping removes a lot of the distinctions made by the original ontology. It is not a surprize that such a mapping is possible, but the purpose of this mapping is to make it possible to determine whether and how the temporal abstraction mechanisms can also be mapped onto the KADS inferences.

4.4 Mapping the TA mechanisms onto KADS inferences

In the previous section we have shown how the temporal–abstraction ontology maps onto the KADS base–ontology. Below we will reformulate the temporal abstraction mechanisms in terms of the formalized KADS inferences as specified in [Aben, 1993a]. In the mappings, we will in particular refer to the inferences **compute**, **assign**, **generalize**. These inferences will be defined when they are introduced.

Point Temporal Abstraction

Let us first formalize *state abstraction*. State abstraction abstracts an input state onto an abstract state. In [Aben, 1993b] we have formalized the KADS **abstract** inference as:

$$c, c' : concept, n : integer; f' : attribute, \varrho_{abstr} : value_relation \quad (1)$$
$$\textbf{abstract}(c, f', \varrho_{abstr}, c') \Leftrightarrow \quad (2)$$
$$\forall_{1 \leq i \leq n}\ F_i(c) \downarrow\ \wedge\ \varrho_{abstr}(\varrho_1(F_1(c), \ldots, F_n(c)), \varrho_2(f'(c))) \rightarrow \varrho_2(f'(c')) \quad (3)$$

This definition states that for all attributes F_i on c on which some value relation ϱ_1 holds, and for all abstraction relations that specify how that value relation can be abstracted to some value relation ϱ_2 on some abstract attribute F', the value relation ϱ_2 is enforced on the output concept. Note that both ϱ_1 and ϱ_2 are universally quantified.

This caters for abstraction rules such as:

$$\varrho_{abstr}(temperature(patient) = 39, fever(patient) = true) \quad (4)$$
$$\varrho_{abstr}(temperature(patient) > 37.5, fever(patient) = true) \quad (5)$$
$$\varrho_{abstr}(temperature(patient) < 37.5, fever(patient) = false) \quad (6)$$

The **state_abstract** inference is a refinement of **abstract**. In this case ϱ_{abstr} is the actual abstraction relation specified at the domain–layer and there is an extra parameter ξ, representing the context information. Hence we get:

$$\textbf{state_abstract}(\sigma, \{\pi_1, \ldots, \pi_n\}, \rho, \xi, \sigma') \Leftrightarrow \quad (7)$$
$$\forall_{1 \leq i \leq n} \quad (8)$$
$$\pi_i(\sigma) \downarrow\ \wedge \varrho_{abstr}(\varrho_1(\xi, \pi_1(\sigma), \ldots, \pi_n(\sigma)), \varrho_2(\pi'(\sigma))) \rightarrow \quad (9)$$
$$\varrho_2(\pi'(\sigma')) \quad (10)$$

Where the single–parameter is a trivial variant of this, with $n = 1$.

If the parameter does not have the same value on all the input states, we might want to replace **state_abstract** with either **rate_abstract** or **gradient_abstract** (see below).

As described above, *classification point temporal abstraction* consists of two steps: *generalization* of the set of input states into a single point–state and *abstracting* the point–state with respect to a given parameter, yielding an abstract point–state:

$$\textbf{classification_pta}(\Sigma, \pi, \rho, \xi, \sigma') \Leftrightarrow \quad (11)$$
$$\textbf{generalize}(\Sigma, \rho, \xi, \sigma) \wedge \quad (12)$$
$$\textbf{state_abstract}(\sigma, \{\pi\}, \rho, \xi, \sigma') \quad (13)$$

The **generalize** inference is defined as follows:

> *In generalization one focusses on common features in the set of instances and tries either to map these on an existing concept, or to develop a new concept. The former resembles* **classify** *[...] The latter is generally known as* induction, *for which various methods have been developed, in particular in machine learning (see e.g. [Charniak & McDermott, 1985]).* [Breuker *et al.*, 1987a; page 37]

Generalize is formalized as:

$$S : instance_set, c : concept, \leq_{subs} : predicate \tag{14}$$
$$\textbf{generalize}(S, c, \leq_{subs}) \Leftrightarrow \tag{15}$$
$$\iota \in S \rightarrow \tag{16}$$
$$F(\iota) \downarrow \leftrightarrow F(c) \downarrow \ \wedge F(\iota) \leq_{subs} F(c) \tag{17}$$

The formalization states that each attribute defined on *all* of the input instances is also defined on the output concept, and the value of this attribute on the output concept is more or equally general as all the values of the attribute on the input instances.

The variant of **generalize** used here is a specialization of this. In this case, the input–set is a set of *states*, the output concept is a state instance and the generalization relation is defined by ρ and the context ξ. **Generalize** would typically rely on queries to the domain layer, where the temporal equivalence relation ρ is defined.

$$\textbf{generalize}(\Sigma, \rho, \xi, \sigma) \Leftrightarrow \tag{18}$$
$$\forall_{\sigma' \in \Sigma} \tag{19}$$
$$\rho(\xi, \tau_s(\sigma'), \tau_s(\sigma)) \wedge \tag{20}$$
$$\rho(\xi, \tau_e(\sigma'), \tau_e(\sigma)) \tag{21}$$

Computational point temporal abstraction is equivalent to classification point temporal abstraction, except that it involves additional computation to map the *set* of input parameters onto a single parameter which is subsequently abstracted. This computation is performed by the **compute** inference:

> *On the basis of some structure of concepts and their instances the value of some dependent factor is calculated. [...] In general, this [inference] refers to a specific set of concepts as input/output [roles]: variables, factors, parameters, etc. By computation some concept may get instantiated as well or simply the value of some instantiated concept may change. Note that the latter implies an* **assign** *[inference] [...].* [Breuker *et al.*, 1987a; page 36]

This can be formalized as:

$$\iota, \iota' : instance, f : attribute \tag{22}$$
$$\textbf{compute}(\iota, f, \iota') \tag{23}$$
$$\exists V : value \tag{24}$$
$$\textbf{assign}(\iota, f, V, \iota') \tag{25}$$

This formalization is a direct transcription of the verbal descriptions. The actual computation is not described, but is hidden in the existential quantification of V. **compute** is a mapping between structures, resulting in the assignement of an attribute-value of an instance. Each specific instance of compute would be formalized such, that the actual relation between the assigned attribute and the structure becomes clear.

Although **compute** can be abused as a kind of *catch–all* operator, in this case the inference is appropriate. The relations and functions that determine the actual computations are domain–specific and hence are defined at the domain layer. **Compute** does not do much more than simply applying these relations and functions. The **compute**

inferences below are parametrized over the actual relations that are used to compute the resulting attribute value.

The **assign** inference is defined as follows:

> *[Inferences] of this type derive values for attributes of concepts and assign them to those attributes. An example is the use of default values. There are subtle variations on this theme. If the attribute had already a value, this value is overwritten, but it does not seem of practical relevance to distinguish also "change value" and "remove value".* [Breuker *et al.*, 1987a; page 36]

$$c, c' : concept, f : attribute \tag{26}$$
$$\textbf{assign}(\iota, f, V, \iota') \Leftrightarrow \tag{27}$$
$$f(\iota') = V \tag{28}$$

This formalization is quite trivial. It states that some attribute is defined on the input instance, and has some value on the output instance. Note that requiring that the attribute is already defined or not, could also be seen as a *precondition* for application of the inference action.

Hence *computational transformation point temporal abstraction* is defined as:

$$\textbf{computational_transformation_pta}(\Sigma, \Pi, \rho, \xi, \sigma') \Leftrightarrow \tag{29}$$
$$\textbf{generalize}(\Sigma, \rho, \xi, \sigma) \wedge \tag{30}$$
$$\textbf{compute}(\Sigma, \Pi, \xi, \pi(\sigma)) \wedge \tag{31}$$
$$\textbf{state_abstract}(\sigma, \{\pi\}, \rho, \xi, \sigma') \tag{32}$$

The parameters in Π are projected onto one parameter π. One could parametrize **computational_transformation_pta** over this π. In the **classification_pta**, it is assumed that the attribute π gets a new value, but one could also decide to introduce a new parameter *abstract_*π containing the abstract value of π.

Preconditions enforce that the input states are *time–points*:

$$\textbf{pre_pta}(\Sigma) \tag{33}$$
$$\forall_{\sigma \in \Sigma} \tag{34}$$
$$\tau_s(\sigma) = \tau_e(\sigma) \tag{35}$$

Temporal Inference

Temporal inference is a KADS **compute** inference The $\otimes$ operator *joins* the input intervals into an output interval, but there could also be a more elaborate operator, which extracts a sub–interval or generalizes to a super–interval:

$$\textbf{temporal_inference_pta}(\Sigma, \pi, \rho, \otimes, \xi, \sigma) \Leftrightarrow \tag{36}$$
$$\textbf{compute}(\Sigma, \pi, \rho, \xi, \otimes, \sigma) \tag{37}$$

The **compute** inference computes the output interval–state and (optionally) the output–value of π. If we have also a time–interval as input, then this is simply **assign**–ed to the output state, where **compute** only (optionally) computes a new value for π:

$$\textbf{temporal_inference_pta}(\Sigma, [\tau_1, \tau_2], \pi, \rho, \otimes, \xi, \sigma) \Leftrightarrow \quad (38)$$
$$\quad \textbf{compute}(\Sigma, \pi, \rho, \xi, \sigma') \wedge \quad (39)$$
$$\quad \textbf{assign}(\sigma', \tau_s, \tau_1, \sigma'') \wedge \quad (40)$$
$$\quad \textbf{assign}(\sigma'', \tau_e, \tau_2, \sigma) \quad (41)$$

The preconditions model the applicability criteria, and the postconditions model the inferential properties ϕ (downward-hereditary, ...), since these should hold when the inference is applied.

$$\textbf{pre_downward_hereditary}(\sigma) \leftrightarrow \quad (42)$$
$$\quad \tau_s(\sigma) < \tau_e(\sigma) \quad (43)$$

I.e. σ must be a interval–state.

$$\textbf{post_downward_hereditary}(\sigma, \sigma') \leftrightarrow \quad (44)$$
$$\quad \tau_s(\sigma) < \tau_s(\sigma') \wedge \quad (45)$$
$$\quad \tau_e(\sigma) > \tau_e(\sigma') \quad (46)$$

I.e. the output state state is a sub–interval of the input state. Note that this requires that the input set to **temporal_inference_pta** contains only one state.

$$\textbf{pre_concatenable}([\sigma_1, \ldots, \sigma_n]) \leftrightarrow \quad (47)$$
$$\quad \forall_{1 \leq i \leq n-1} \quad (48)$$
$$\quad\quad \tau_s(\sigma_i) < \tau_e(\sigma_i) \wedge \quad (49)$$
$$\quad\quad \tau_s(\sigma_i) \leq \tau_e(\sigma_{i+1}) \wedge \quad (50)$$
$$\quad\quad \tau_s(\sigma_{i+1}) - \tau_e(\sigma_i) \leq 1 \quad (51)$$

I.e. the input states should be ordered intervals–states, and there should be no time–gaps between each succesive input state.

$$\textbf{post_concatenable}([\sigma_1, \ldots, \sigma_n], \sigma') \leftrightarrow \quad (52)$$
$$\quad\quad \tau_s(\sigma_1) = \tau_s(\sigma') \wedge \quad (53)$$
$$\quad \tau_e(\sigma_n) = \tau_e(\sigma') \quad (54)$$

I.e. the output state state is a super–interval of the input state.

$$\textbf{pre_upward_hereditary}([\sigma_1, \ldots, \sigma_n]) \leftrightarrow \quad (55)$$
$$\quad \forall_{1 \leq i \leq n-1} \quad (56)$$
$$\quad\quad \tau_s(\sigma_i) < \tau_e(\sigma_i) \wedge \quad (57)$$
$$\quad\quad \tau_s(\sigma_i) \leq \tau_e(\sigma_{i+1}) \quad (58)$$

I.e. the input states should be ordered intervals–states.

post_upward_hereditary$([\sigma_1, \ldots, \sigma_n], \sigma') \leftrightarrow$ (59)

$\tau_s(\sigma_1) > \tau_s(\sigma') \wedge$ (60)

$\tau_e(\sigma_n) < \tau_e(\sigma')$ (61)

I.e. the output state state is a super–interval of the input state.

Note that the postconditions defined here do not *necessarily follow* from the definition of the **temporal_inference_pta** inference. They *should* necessarily follow from the inference given a particular time–interval and/or a particular computation rule at the domain layer.

Temporal Interpolation

State abstraction is similar to *state–abstract* as defined above (lines (7–10)), with on the domain layer such relations as:

$$abs_state(WBC_count(\sigma) = 0.5, WBC_count(\sigma) = low)) \tag{62}$$

i.e., $\varrho_{abstr} = abs_state, \varrho_1 = "=", \pi = WBC_count, \varrho_2 = "=", \pi' = WBC_count$ and ξ is not specified.

Note that here we take $\pi' = \pi$, but one can imagine circumstances where this is not the case, e.g. when abstracting a given temperature to a symptom.

Gradient and rate abstraction are similar to *state abstraction* as well but here we abstract respectively the *magnitude* and *rate change* in the parameter value. It would be more appropriate to separate this out into two steps: a **generalize** inference that generalizes the set of states into a single abstract state, and a **compute** inference that computes and assigns the value of the (rate of) change in the parameter to a new parameter on the abstract state.

gradient_abstract$([\sigma_1, \ldots, \sigma_n], \pi, \sigma') \Leftrightarrow$ (63)

generalize$([\sigma_1, \ldots, \sigma_n], \sigma'') \wedge$ (64)

compute$(\sigma'', \pi_{grad}, \delta_{grad}(\pi(\sigma_1), \pi(\sigma_n), \tau_s(\sigma_1), \tau_e(\sigma_n)), \sigma')$

δ_{grad} is a domain specific function that is used to compute the change in value of π. Hence δ_{grad} would be defined at the domain–layer, and returns a nominal value. This could be specified as a postcondition:

post_grad_abstr$(\Sigma, \pi, \sigma') \leftrightarrow \pi_{grad}(\sigma') \in \{$`decreasing, same,`$\ldots\}$ (65)

Similarly *rate abstraction* can be defined:

rate_abstract$([\sigma_1, \ldots, \sigma_n], \pi, \sigma') \Leftrightarrow$ (66)

gradient_abstract$([\sigma_1, \ldots, \sigma_n], \pi, \sigma'') \wedge$ (67)

compute$(\sigma'', \pi_{rate}, \delta_{rate}(\pi(\sigma_1), \pi(\sigma_n), \tau_s(\sigma_1), \tau_e(\sigma_n)), \sigma')$

In this case, the postcondition would be:

$$\textbf{post_rate_abstr}(\Sigma, \pi, \sigma') \leftrightarrow \pi_{rate}(\sigma') \in \{\texttt{zero, slow, fast,...}\} \quad (68)$$

Note that in these cases since the actual computation is done at the domain layer, it is hard to distinguish between *rate abstraction* and *gradient abstraction* at the inference layer, other than by looking at the post–conditions, which could also be formulated as "semantic" constraints on the domain–computations.

Temporal interpolation is now defined as a conjunction of one of state–, gradient– or rate–abstraction and one of primary– or secondary interpolation. Primary interpolation can be formalized as:

$$\textbf{primary_interpolation}(\sigma_1, \sigma_2, \Delta, \pi, \theta, \xi, \sigma') \Leftrightarrow \quad (69)$$
$$(\tau(\sigma_2) - \tau(\sigma_1)) \leq \Delta(\pi, \theta, \pi(\sigma'), 0, 0, \xi) \rightarrow \quad (70)$$
$$\tau_s(\sigma') = \tau(\sigma_1) \wedge \quad (71)$$
$$\tau_e(\sigma') = \tau(\sigma_2) \quad (72)$$

With preconditions:

$$\textbf{pre_primary_interpolation}(\sigma_1, \sigma_2) \Leftrightarrow \quad (73)$$
$$\tau_s(\sigma_1) = \tau_e(\sigma_1) \wedge \quad (74)$$
$$\tau_s(\sigma_2) = \tau_e(\sigma_2) \wedge \quad (75)$$
$$\tau(\sigma_1) \leq \tau(\sigma_2) \quad (76)$$

And secondary interpolation can be formalized as (a generalization of the former):

$$\textbf{secondary_interpolation}(\sigma_1, \sigma_2, \Delta, \pi, \theta, \xi, \sigma') \Leftrightarrow \quad (77)$$
$$\tau_s(\sigma_2) - \tau_e(\sigma_1) \leq \Delta(\pi, \theta, \pi(\sigma'), \tau_e(\sigma_1) - \tau_s(\sigma_1), \tau_e(\sigma_2) - \tau_s(\sigma_2), \xi) \rightarrow \quad (78)$$
$$\tau_s(\sigma') = \tau_s(\sigma_1) \wedge \quad (79)$$
$$\tau_e(\sigma') = \tau_e(\sigma_2) \quad (80)$$

With preconditions:

$$\textbf{pre_secondary_interpolation}(\sigma_1, \sigma_2) \Leftrightarrow \quad (81)$$
$$\tau_s(\sigma_1) < \tau_e(\sigma_1) \wedge \quad (82)$$
$$\tau_s(\sigma_2) < \tau_e(\sigma_2) \wedge \quad (83)$$
$$\tau_s(\sigma_1) \leq \tau_e(\sigma_2) \quad (84)$$

which is a generalization of the *concatenable* precondition in lines (47–50).

5 Conclusions

In this paper we have presented a mapping of elements of the PROTÉGÉ–II framework onto elements of the KADS framework for knowledge based systems design. We want to stress that the mapping is not a complete one. We for instance have left out the mapping of the control knowledge in the PROTÉGÉ–II approach onto KADS task layers. Also, the mapping of the domain knowledge is restricted here only to the mapping of the (abstract) domain ontologies. The reason for these restrictions is twofold: it would be hard to

show the complete mapping in the space available and we wanted to stress the mapping between the most central aspects of the approaches, i.e. between the *mechanisms* and the *inferences*.

The following observations can be made as a result of this exercise:

- The mapping between ontologies was feasible
- This mapping removes a lot of detail
- The mechanisms do not map onto a single inference, this exemplifies the difference in grainsize of PROTÉGÉ–II mechanisms and KADS inferences
- The nature of the temporal abstraction is more closely related to KADS **generalize** and **compute** inferences than to KADS **abstract**. This exemplifies that the KADS names do not always coincide with their interpretations in different tasks.
- The mechanisms were all expressible in KADS inferences, showing that the set seems adequate for this type of tasks.
- The relation between the inferences, and hence the mechanisms became more clear, revealing interrelations that were not explicit before.

  For example, if we compare the preconditions of **secondary_interpolation** with those of **temporal_inference_pta**, we can make the following observations. The precondition of **secondary_interpolation** is more general than the precondition of the *concatenable* property of **temporal_inference_pta**, equivalent to the precondition of the *upward_hereditary* property, and more specific than the precondition of the *downward_hereditary* property. This implies for instance that whenever the *downward_hereditary* precondition holds, we can either perform **secondary_interpolation** or **temporal_inference_pta**.

References

Aben, M. (1993a). CommonKADS inferences. ESPRIT Project P5248 KADS-II/ M2/ TR/ UvA/ 041/ 1.0, University of Amsterdam.

Aben, M. (1993b). Formally specifying re–usable knowledge model components. *Knowledge Acquisition*, 5(2). To Appear.

Allen, J. (1984). Towards a general theory of action and time. *Artificial Intelligence*, 23:123–154.

Benjamins, V. R. (1993). *Problem Solving Methods for Diagnosis.* PhD thesis, University of Amsterdam, Amsterdam, The Netherlands.

Brachman, R. J. & Schmolze, J. G. (1985). An overview of the KL-ONE knowledge representation system. *Cognitive Science*, 9:171–216.

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. (1987a). Model Driven Knowledge Acquisition: Interpretation Models. ESPRIT Project P1098 Deliverable D1 (task A1), University of Amsterdam and STL Ltd.

Breuker, J. A., Winkels, R. G. F., & Sandberg, J. A. C. (1987b). A shell for intelligent help systems. In *Proceedings of the 10th IJCAI*, pages 167–173, Milan, Italy.

Chandrasekaran, B. (1986). Generic tasks in knowledge-based reasoning: High level building blocks for expert system design. *IEEE Expert*, 1.

Charniak, E. & McDermott, D. (1985). *Introduction to Artificial Intelligence*. Addison Wesley, Mass.

Clancey, W. J. (1992). Model construction operators. *Artificial Intelligence*, 53(1):1–115.

Genesereth, M. R. & Fikes, R. E. (1992). Knowledge interchange format version 3.0 reference manual. Report Logic 92-1, Logic Group, Stanford University, California.

Gruber, T. R. (1993). A translation approach to portable ontology specifications. *KAJ*, 5(2). To Appear.

McCarthy, J. (1986). Applications of circumscription to formalizing conmmonsense knowledge. *Artificial Intelligence*, 28:89–116.

McDermott, D. (1982). A temporal logic for reasoning about processes and plans. *Cognitive Science*, 6:101–155.

Neches, R., Fikes, R. E., Finin, T., Gruber, T. R., Patil, R. S., Senator, T., & Swartout, W. S. (1991). Enabling technology for knowledge sharing. *The AI Magazine*, 12(3):16–36.

Patil, R. S., Fikes, R. E., Patel-Schneider, P. F., McKay, D., Finin, T., Gruber, T. R., & Neches, R. (1992). The DARPA knowledge sharing effort: Progress report. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Third International Conference*. Morgan Kaufmann.

Schreiber, A. T. (1992). The KADS approach to knowledge engineering. editorial special issue. *Knowledge Acquisition*, 4(1):1–4.

Schreiber, A. T., van Heijst, G., Lanzola, G., & Stefanelli, M. (1993). Knowledge organisation in medical KBS construction. submitted to AIME–93.

Shahar, Y., Aben, M., & Musen, M. (1993). A framework for acquiring temporal abstraction knowledge. Knowledge Systems Laboratory report KSL-93-02 (Nr. 459), Knowledge Systems Laboratory, Stanford University, Stanford, CA, USA.

Shahar, Y. & Musen, M. (1993). RÉSUMÉ: A temporal–abstraction system for patient monitoring. *Computers and Biomedical Research*, 26:255–273.

Shahar, Y., Tu, S. W., & Musen, M. A. (1992). Knowledge acquisition for temporal–abstraction mechanisms. *Knowledge Acquisition*, 4:217–236.

Shoham, Y. (1987). Temporal logics in AI: Semantical and ontological considerations. *Artificial Intelligence*, 33:89–105.

Steels, L. (1990). Components of expertise. *AI Magazine*. Also as: AI Memo 88-16, AI Lab, Free University of Brussels.

van Heijst, G., Lanzola, G., Schreiber, A. T., & Stefanelli, M. (1993). Methodological foundations of medical KBS construction. Technical Report GAMES-II/T1.1/UvA/PP/009/1.0, University of Amsterdam and University of Pavia. Available from: University of Amsterdam, Social Science Informatics, Roetersstraat 15, NL-1018 WB, Amsterdam, The Netherlands.

Wielinga, B. J., Schreiber, A. T., & Breuker, J. A. (1992). KADS: A modelling approach to knowledge engineering. *Knowledge Acquisition*, 4(1):5–53. Special issue 'The KADS approach to knowledge engineering'. Reprinted in: Buchanan, B. and Wilkins, D. editors (1992), *Readings in Knowledge Acquisition and Learning*, San Mateo, California, Morgan Kaufmann, pp. 92-116.