# Temporal Semantics of Compositional Task Models and Problem Solving Methods

Frances Brazier, Jan Treur, Niek Wijngaards and Mark Willems

Vrije Universiteit Amsterdam, Department of Artificial Intelligence
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands
Email: {frances, treur, niek}@cs.vu.nl
URL: http://www.cs.vu.nl/~wai

**Keywords:** compositional, task model, problem solving method, dynamics, temporal semantics, knowledge-based systems.

**Abstract** Task models and problem solving methods can be specified informally or formally. In recent years various approaches have formalized their notion of task model or problem solving method. Most modelling approaches concentrate on the form of a task model or problem solving method rather than on their precise semantics; a formalisation is often only a syntactical formalisation. A more precise definition of the semantics requires explication of the control of a system's behaviour. In this paper temporal semantics is defined for a compositional modelling approach to task models and problem solving methods. The semantics is a description of a compositional system's behaviour; a temporal approach provides a means to describe the dynamics involved. The formalisation of the semantics is based on compositional three-valued temporal models. The compositional structure of information states, transitions and reasoning traces provides a transparant model of the system's behaviour, both conceptually and formally.

## 1 INTRODUCTION

Complex tasks in which reasoning plays an important role, such as design or diagnostic tasks, are most often extremely dynamic. The aim in knowledge engineering is to model such complex behaviour. The common approach is to model complex functionality by means of task composition; a complex task is composed of a number of smaller, less complex tasks. Most knowledge engineering approaches take such a task oriented approach; e.g., MIKE/KARL [3], [4], 16], CommonKADS [33], VITAL [34], PROTÉGÉ-II [30], TASK [28] and MILORD [1]. Within knowledge engineering the ability to formally describe the behaviour of models (of complex tasks) contributes to modelling, design, evaluation, maintenance, validation and verification, and reuse of models [17], [37]. As such, formal methods provide a means to formally describe a complex task thereby providing a common ground for informal models of the same complex task.

A framework that allows conceptual and formal specification of such models should be powerful enough to capture such behaviour in an explicit and transparent manner. Early work on formalisation of task models and problem solving methods can be found in [5], [24], [38]. These formalisations only address the syntax of the models. A number of the current approaches to modelling complex reasoning tasks, in which formal specification plays an important role, are MIKE/KARL [3], [16], CommonKADS/(ML)$^2$ [2], [19], VITAL [34], TASK [28] and MILORD [1]. Also these approaches include a formal syntax for the specification language, but the formal semantics of such systems from a dynamic perspective are often not defined in detail; for a description that *is* detailed, see [16].

Both a conceptual and formal definition of the temporal semantics of behaviour is especially of importance for dynamic tasks. Formal semantics not only provides a basis for validation and verification of system behaviour, but specifications of components at different levels of abstraction with well-defined semantics also provide a means to enable automated support for re-design of task models and components. Automated support for re-design, for example, often requires explicit formulation of requirements on system behaviour, based on well-defined semantics. The temporal approach to semantics described in this paper assumes sequential processing, as do all other approaches to modelling of tasks and problem solving methods mentioned above (in contrast to multi-agent systems, where parallel processing is assumed; e.g., [7]).

Temporal modelling and temporal reasoning is often applied in the context of reasoning about a dynamic world (e.g. patient data) cf. [35]. Another application of temporal modelling is reasoning about the order in which certain operators need to be applied, cf. [13], [14]. In contrast this paper addresses temporal semantics of reasoning tasks themselves.

Section 2 provides an overview of the knowledge modelled and specified in task models. In Section 3 the perspective on temporal semantics is introduced and the basic concepts are defined. In Section 4 definitions of the concepts required to formalise compositional information states are introduced, followed by definitions of concepts directly related to transitions between (hierarchical) component information states in Section 5. The formalisation of the resulting compositional behaviour is defined in terms of the temporal approach in Section 5 and discussed in Section 6. Further discussion of applications for which this approach has been successfully applied extends beyond the scope of this paper: see, for example [21].

## 2 COMPOSITIONAL MODELLING OF TASKS AND PROBLEM SOLVING METHODS

Using the development method DESIRE problem solving behaviour in complex tasks in knowledge-intensive domains is modelled explicitly in a compositional manner. The resulting products are specifications of reflective (knowledge based) compositional architectures (including task and domain knowledge) and specifications of problem description and design rationale. To this end a problem description, a design rationale, and three levels of design are distinguished: conceptual design, detailed design, operational design, as shown in Figure 1.

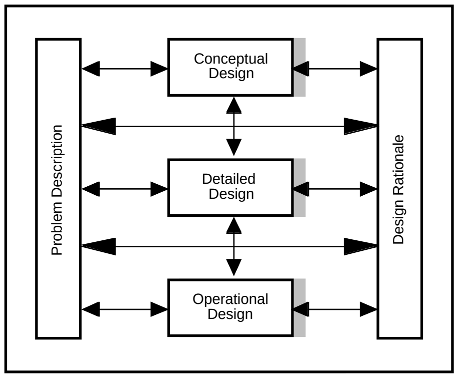


**Fig 1.** Problem description, levels of design and design rationale

The relations between these levels of design are well defined. The detailed specification, for example, always preserves the structures defined at the conceptual level of specification. However, at the detailed level of design, additional structures (and details) are added. At the level of operational design the formal semantics of the level of detailed design are ‘operationalised’. It is not necessary for an operational design to have a structure preserving relationship with the detailed level of design; although structure preservation enhances transparancy, prototyping and tracing. Current prototyping tools preserve structure between detailed design and operational design. The design rationale contains all decisions, underlying assumptions, reasons, etc., upon which the conceptual, detailed and operational design are based.

To acquire specifications of complex (reasoning) systems extensive interaction between knowledge engineers and experts is required. The purpose of such interaction, within DESIRE, is to acquire a *shared* (agreed) *task model*: a model on which both the knowledge engineers and the experts agree [9]. Existing compositional task models, usually generic task models, are most often used to initially structure knowledge acquisition. Which models are used, depends on the initial description of a task or task components: in interaction with one or more experts existing models are examined, discussed, rejected, modified, refined and/or instantiated. Compositional generic task models provide a means to specify *problem solving methods* (independent of domain onotologies and domain knowledge). Such compositional task models are *generic* in two senses: they are a description of the problem solving method used in the task both at an abstract level and application domain independent. Initial abstract descriptions of tasks can be used to generate a variety of more specific task descriptions through refinement and composition (existing descriptions can be employed) in interaction with experts.

During knowledge acquisition, knowledge of the application domain itself is also acquired: such application specific knowledge is modelled independently in knowledge structures, and is included in task models by reference to such structures. Knowledge structures are also shared models: models of the domain. Which techniques are used for knowledge elicitation is not predefined. Techniques vary in their applicability, depending on the situation, the resources, the task, the type of knowledge on which the knowledge engineer wishes to focus, etc.

A shared task model is, in fact, a mediating model [18]. It mediates between a knowledge engineer and an expert, but also between a knowledge engineer and system design. Within DESIRE task models provide the basis for system architecture. Tasks are mapped onto components, interactions between tasks are mapped onto information links between components. Domain knowledge is mapped onto structures which are included and referenced in specifications of components. The goals pursued and the roles of the parties involved in relation to the goals are defined. A description of the situation in which such goals can be pursued, and of the assumptions with respect to task performance are made explicit.

The result of  analysis is a conceptual specification of knowledge of:

1. *process composition*
   - identification of tasks at different abstraction levels
   - task delegation
   - information exchange between tasks
   - task sequencing
2. *knowledge composition*
   - information types
   - knowledge bases
3. *relation between process composition and knowledge composition*

One of the tasks distinguished in a task model for elevator configuration [8] is used to illustrate the conceptual representation of these types of knowledge.

## 2.1 Identification of tasks at different levels of abstraction

Identification of tasks and their abstraction levels includes knowledge of a task hierarchy, knowledge of the types of information required as input and resulting as output, and knowledge of the reflective nature of tasks with respect to other tasks.

A *task hierarchy* defines the tasks of which a task is composed and the relations between tasks. A one-to-many relation between tasks can, for example, be specified by means of a table (not shown) and depicted as a tree structure or a box-in-box structure (see Figures 2 and 3).

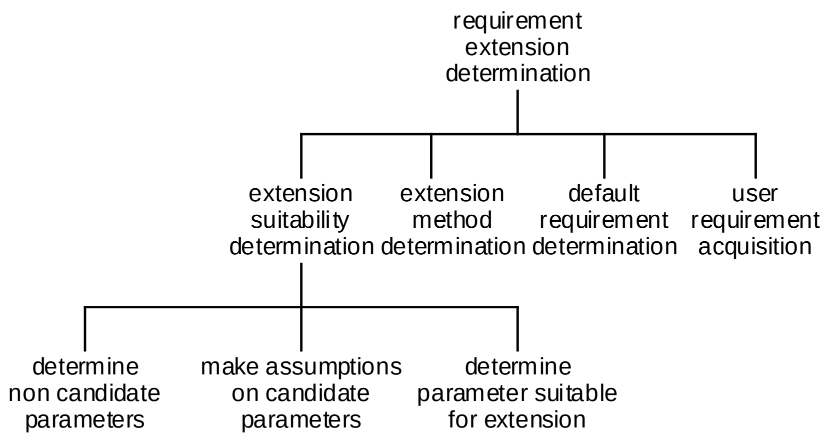

requirement extension determination
extension suitability determination
extension method determination
default requirement determination
user requirement acquisition
determine non candidate parameters
make assumptions on candidate parameters
determine parameter suitable for extension

**Fig 2.** Pictorial specifications of a task composition.

The task ‘requirement extension determination’ depicted in Figure 2, one of the tasks in the elevator configuration task [8] is the task responsible for proposing extensions to a given set of requirement parameters. To this purpose ‘extension suitability determination’ determines which requirement parameters are best suited to be given a value, ‘extension method determination’ decides which method to use to determine a value for the chosen requirement parameter (by, for example, default reasoning or user consultation), ‘default requirement determination’ assigns a value to the chosen requirement parameter, and ‘user requirement acquisition’ obtains a value by interacting with the user, and assigns the value to the chosen parameter.

The task of ‘extension suitability determination’ is composed of three tasks, of which the task ‘determine non-candidate parameters’ derives which parameters are not suitable candidate parameters, the task ‘make assumptions on candidate parameters’ employs a closed-world-assumption to determine which candidate parameters are still eligible for extension, and the task ‘determine parameter suitable for extension’ selects one of these eligible parameters as the suitable parameter. The task ‘extension suitability determination’ as a whole is non-monotonic: it does not conserve its output when more input is available (not retracting already known input). Each of the tasks of which the task ‘extension suitability determination’ is composed, however, is monotonic.

Conceptually, for each task, the *types of information* required as *input* or generated as *output* of a task, are specified. Names are defined for information types (names chosen by the knowledge engineer and the expert(s)) and relations expressing how such information is related to a tasks’ output and/or input. In a pictorial representation, each task can be annotated with information regarding the types of information of its input and output, for example, with input to the left and output to the right as shown in Figure 3.

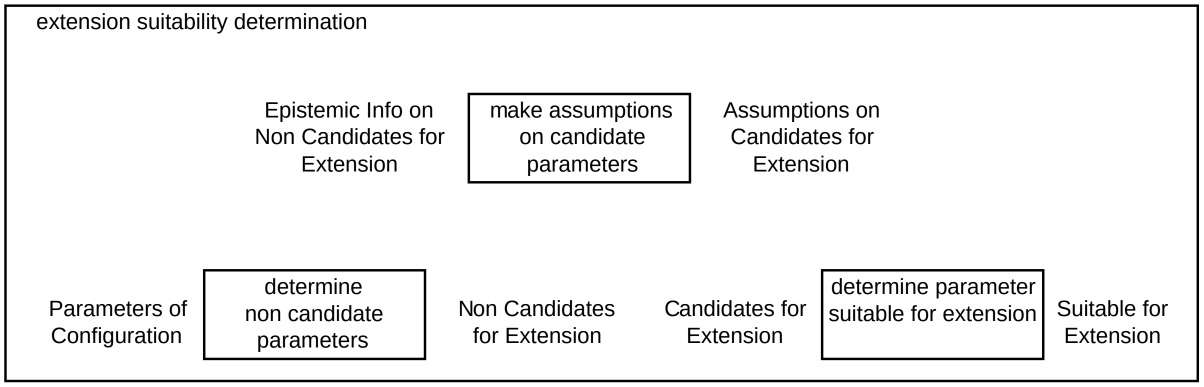

extension suitability determination
Epistemic Info on Non Candidates for Extension
make assumptions on candidate parameters
Assumptions on Candidates for Extension
Parameters of Configuration
determine non candidate parameters
Non Candidates for Extension
Candidates for Extension
determine parameter suitable for extension
Suitable for Extension

**Fig 3.** Input and output information types of task ‘extension suitability determination’.

The *reflective nature of tasks* is another important element of a task composition emphasised in DESIRE. A clear distinction between object-level reasoning about a domain and meta-level reasoning about the state and goals of a system is essential for a transparent specification of a system. This object-meta distinction between tasks can be specified explicitly as an object-meta relation, an example is shown in Table 1.

| *object* | *with respect to* | *meta* |
|---|---|---|
| determine non candidate parameters | | make assumptions on candidate parameters |
| determine parameters suitable for extension | | make assumptions on candidate parameters |

**Table 1.** Object-meta distinction between tasks.

## 2.2 Information exchange between tasks

Knowledge of information exchange between (sub-)tasks defines the types of information transferred between tasks. More specifically, the relations expressing information exchange between tasks are explicitly specified and named, in particular to control the information flow (see Section 2.3).

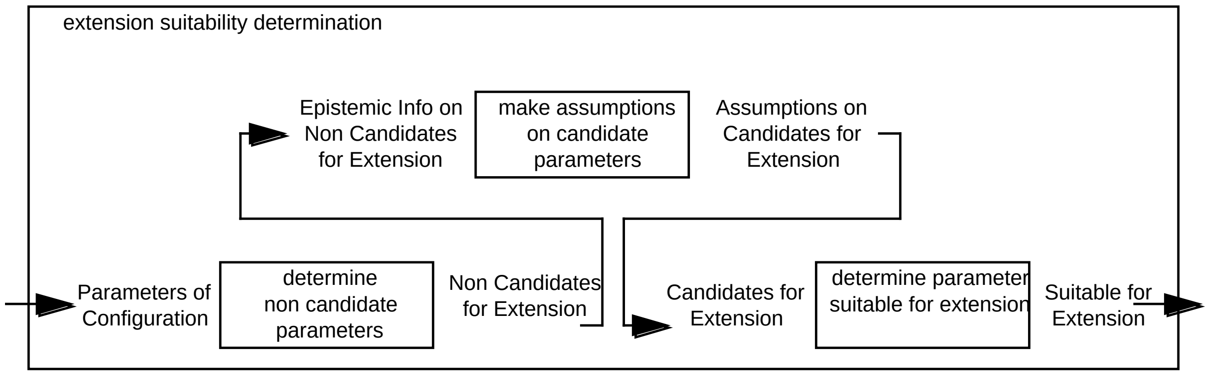

extension suitability determination
Epistemic Info on Non Candidates for Extension
make assumptions on candidate parameters
Assumptions on Candidates for Extension
Parameters of Configuration
determine non candidate parameters
Non Candidates for Extension
Candidates for Extension
determine parameter suitable for extension
Suitable for Extension

**Fig 4.** Information exchange among sub-tasks and between sub-tasks and the task of 'extension suitability determination'.

In Figure 4 examples of two kinds of relations are shown in a pictorial representation: private information exchange between the output information type of one sub-task and the input information type of another sub-task, and mediating information exchange between the output information type of a sub-task and the output information type of its task (and vice versa).

Not only are the types of information to be transferred from one task to the next defined, but also the grounds upon which the 'decision' to forward this information are based. Explicit evaluation criteria, specified for this purpose are discussed in the next section.

### 2.3 Task sequencing

Knowledge of sequencing of task and information exchange defines temporal relations between tasks (and information transfer): which tasks must (directly) precede other tasks and which information exchange is required. Task sequencing knowledge specifies under which conditions which tasks and information links are activated. These conditions, preconditions for task activation, may, for example, include *evaluation criteria* expressed in terms of the evaluation of the results (success or failure) of one or more of the preceding tasks. The evaluation criteria, the result and the name of the next task(s) to be activated can be specified in an (incomplete) pictorial representation where 'control' arrows are labelled with the evaluation criteria (**–** for failure and **+** for success), as shown in figure 5, for example.

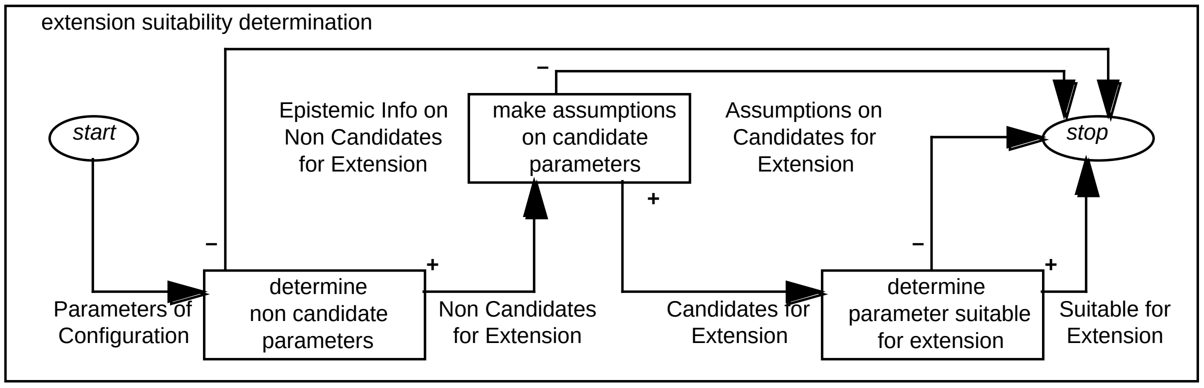

extension suitability determination
start
Epistemic Info on Non Candidates for Extension
make assumptions on candidate parameters
Assumptions on Candidates for Extension
stop
–
+
Parameters of Configuration
determine non candidate parameters
Non Candidates for Extension
Candidates for Extension
determine parameter suitable for extension
Suitable for Extension

**Fig 5.** Pictorial representation of task sequencing in 'extension suitability determination'.

In a detailed specification, task control knowledge is expressed in temporal rules. Current research focuses on suitable alternative representations.

## 2.4 Task delegation

In complex situations often a number of autonomous systems and/or users are involved. Knowledge of task delegation refers to the division of tasks amongst participants: minimally which tasks are to be performed by the system and which by one or more users. In more complex situations often more participants are involved. Task delegation is defined by a set of participants (i.e. agents) and a relation between tasks and sub-sets of the set of agents. See Table 2 for an example in which two agents, the user and the system, are responsible for task performance.

| *task* | *agent(s)* |
|---|---|
| RQS Extension Determination | System, User |
| Extension Suitability Determination | System |
| Extension Method Determination | System |
| Default Requirement Determination | System |
| User Requirement Acquisition | User |
| Determine Non-Candidate Parameters | System |
| Make Assumptions on Candidate Parameters | System |
| Determine Parameters Suitable for Extension | System |

**Table 2.** Task delegation of tasks of 'RQS Extension Determination'.

A pictorial representation can be obtained by placing the (sets of) agents in the tree representing the task hierarchy.

## 2.5 Knowledge structures and relation to tasks

Knowledge of knowledge structures entails specification of additional knowledge needed for task performance: for input and output information structures but also internal knowledge bases used for reasoning. For each task the names of the associated internal information types are shown in an example below.

| *task* | *internal information type* |
|---|---|
| Determine Non-Candidate Parameters | Det-Non-Cand-Param Info Type |
| Make Assumptions on Candidate Parameters | Make-Ass-on-Cand-Param Info Type |
| Determine Parameters Suitable for Extension | Det-Param-Suit-for-Exten Info Type |

**Table 3.** Internal information types for tasks within 'Extension Suitability Determination'.

Information types can be composed of sub-information types as shown in Table 4.

| *information type* | *sub-information type* |
|---|---|
| Det-Non-Cand-Param Info Type | Parameter Dependencies Info Type<br>Parameters in Current RQS Info Typ |

**Table 4.** Hierarchy of information types.

The knowledge bases associated with each task are depicted below in Table 5.

| *task* | *knowledge base* |
|---|---|
| Determine Non-Candidate Parameters | Determine Non-Candidate Parameters Knowledge |
| Make Assumptions on Candidate Parameters | Make Assumptions on Candidate Parameters Knowledge |
| Determine Parameters Suitable for Extension | Determine Parameters Suitable for Extension Knowledge |

**Table 5.** Relation between knowledge bases and tasks within 'Extension Suitability Determination'.

Knowledge bases can be composed of sub-knowledge bases as shown in Table 6 for the knowledge base 'Determine Non-Candidate Parameters Knowledge'.

| *knowledge base* | *sub-knowledge base* |
|---|---|
| Determine Non-Candidate Parameters Knowledge | Parameters in Current RQS Knowledge<br>Parameter Dependency Knowledge |

**Table 6.** Kknowledge bases at different levels of abstraction.

The relations between information types and knowledge bases for tasks within "Extension Suitability Determination" are shown in Table 7.

| *knowledge base* | *information types* |
|---|---|
| Determine Non-Candidate Parameters Knowledge | Parameters of Configuration<br>Det-Non-Cand-Param Info Type<br>Non-Candidates for Extension |
| Make Assumptions on Candidate Parameters | Epistemic info on non-candidate for extension<br>Make-Ass-on-Cand-Param Info Type<br>Assumptions for Candidates for Extension |
| Determine Parameters Suitable for Extension | Candidates for Extension<br>Det-Param-Suit-for-Exten Info Type<br>Suitable for Extension |

**Table 7.** Relation between knowledge bases and information types.

## 2.6 Detailed specification

Specification of a system is most often an iterative process: during knowledge acquisition more detailed knowledge is acquired and modelled. The types of knowledge described during conceptual analysis have a (unique) counterpart in the specification. The types of knowledge become more explicit, and are defined more precisely. Often a conceptual design is adapted during the process of system design, as the result of more detailed analysis of a task.

Each task is mapped onto a component. Each component has a uniform structure (see Figure 6), that distinguishes task control information and kernel information. The kernel information of a primitive component is, in fact, a knowledge base. The kernel of a composed component contains components and information links. Another distinction in the uniform structure of a component is the distinction between public and private information, a distinction which is essential to information hiding.

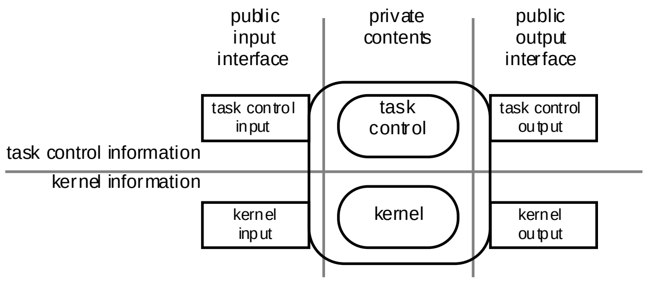


**Fig 6.** Uniform structure of a component.

During detailed specification, (primitive) processes or tasks are mapped onto (primitive) components, information exchange onto information links, and task sequencing knowledge onto task control knowledge. Moreover, the input, output, and internal information types are detailed. This entails:

- extending the process composition to include references to the information types required for input and output, and to include specifications of the meta/object distinctions.
- defining the information links between and within components at the level of the ground atoms within the interfaces of the components.
- specifying the sequencing of component and information link activation and the conditions under which components and links are to be activated.
- defining the information structures and knowledge to which the conceptual description referred: (object and/or meta-level) information types for input and output, and knowledge bases for primitive components.

The detailed specification is reflected in the syntax of DESIRE. Closely related to syntax is the formal semantics that gives it formal meaning. This will be addressed in Sections 3, 4 and 5.

## 2.7 Comparison to other approaches

The formal specification language of DESIRE has been shown in a comparison of specification languages to be more flexible in modelling reasoning patterns [17]. In terms of expressive power, declarativeness, adequacy to specify dynamic aspects of reasoning patterns, possibility to specify multi-level architectures, adequacy to specify non-classical reasoning, executability and availability of formal semantics, the formal specification framework DESIRE is to some extent comparable [10] to other formal specification frameworks such as CommonKADS/(ML)$^2$ [2] VITAL [34], TASK [28], and MIKE/KARL [3], [4]. DESIRE differs from these other approaches in that specifications are executable and agents and integrated systems can be specified. The formal specification languages differ in expressiveness of control knowledge [37], 17], [31].

Task control knowledge within DESIRE is located within each composed task (or component); i.e. distributed control. In (ML)$^2$ and KARL task control knowledge is expressed in a separate layer [31]. Also the ability to perform reflective reasoning (both upward and downward reflection) is a major difference, enhancing DESIRE's flexibility in modelling reasoning patterns.

### 2.8 Example task model

In Section 2.1 part of the task model for elevator configuration [8] that is the task 'requirement extension determination' is described. This task proposes extensions to the current set of requirements (on parameters). In Figure 7 the task model is shown; components and information links are depicted in addition to part of the task control knowledge.

This specific task model is used throughout the remainder of this paper for illustrative purposes.

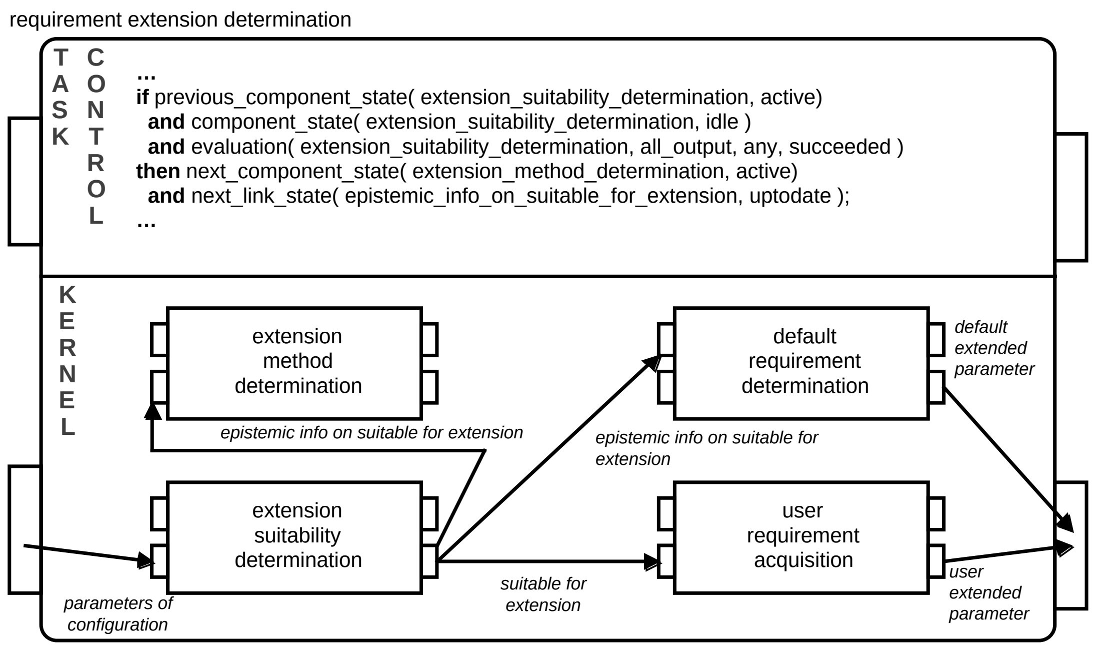

requirement extension determination
TASK CONTROL
...
if previous_component_state( extension_suitability_determination, active)
and component_state( extension_suitability_determination, idle )
and evaluation( extension_suitability_determination, all_output, any, succeeded )
then next_component_state( extension_method_determination, active)
and next_link_state( epistemic_info_on_suitable_for_extension, uptodate );
...
KERNEL
extension method determination
default requirement determination
default extended parameter
epistemic info on suitable for extension
epistemic info on suitable for extension
extension suitability determination
user requirement acquisition
parameters of configuration
suitable for extension
user extended parameter

**Fig. 7.** Example task model of 'requirement extension determination'.

## 3 TEMPORAL SEMANTICS OF REASONING BEHAVIOUR

This section elaborates on the temporal semantics of tasks and problem solving methods. In this paper a state-based semantics is chosen where each component has a three-valued information state. Partial models [6], [27] are used to formalise three-valued information states, representing (incomplete) world descriptions (e.g., [26]). To define formal semantics of reasoning behaviour in (hierarchical) compositional architectures, a recently developed approach based on (partial) temporal models is adopted [12], [20], [36]. Within this approach the semantics of a complex reasoning process is formalised by a set of (alternative) reasoning traces. A reasoning trace is formalised by a partial temporal model, i.e., a sequence of partial models. Behaviour is formalised as sequences of information states in which truth-values are assigned to elements that together describe the domain in which a component reasons. The current state of a component is reflected by the current assignment of truth-values. The behaviour of a compositional architecture is mirrored in its successive (overall) information states, each defined by the composition of the information states of its components.

The elements used to describe the states (the ground atoms) are expressed in a language defined by an information type. Each component within a compositional architecture has an information state describing the truth values (*false* (0), *true* (1) and *undefined* (u)) of atoms of the component.

> **Definition 3.1 (information type, information state)** An *information type* $\Sigma$ is a structure of symbols defining a set of *ground atoms* At($\Sigma$). An *information state* for an information type $\Sigma$ is a mapping from the set of ground atoms At($\Sigma$) to the set of truth values {0, 1, u}; i.e., a *(partial) model* $M : At(\Sigma) \to \{0, 1, u\}$. The set of all information states of information type $\Sigma$ is denoted by IS($\Sigma$).

An example of a structure that defines an information type is a tuple of (sub-)sorts, constants, functions, and predicates of a order-sorted predicate logic, used in the DESIRE specification below, as is the use of referenced information types for importing known information types.

```
information type RequirementParameter
  sorts
    RequirementParameter
  objects
    car_cab_height, … : RequirementParameter;
end information type
```

```
information type NonCandidateReqParm
  information type RequirementParameter;
  relations
    non_candidate: RequirementParameter;
end information type
```

The set of ground atoms defined for NonCandidateReqParm is defined as follows:

< non_candidate(car_cab_height), …, non_candidate(car_phone), …, non_candidate(platform_width) >

Formalising information states as partial models makes it possible to model the reasoning behaviour of common inference mechanisms, such as chaining or unit-resolution, in terms of all ground literal conclusions that have been derived up to a certain moment in time. Information states can be given a structure similar to the structure of the task model. This *composed information state* facilitates the compositional definition of behaviour.

Behaviour is the result of transitions from one information state to another. Transitions are defined within the compositional structure of the information states they manipulate: a transition only changes the information state in one of its components.

> **Definition 3.2 (transition)** A *transition between information states* is a pair of partial models; i.e., an element $< S, S' >$ (also denoted by $S \to S'$) of IS($\Sigma$) x IS($\Sigma$). A *transition relation* is a set of these transitions, i.e. a relation on IS($\Sigma$) x IS($\Sigma$).

If a transition relation is functional then it specifies deterministic behaviour. By applying transitions in succession, sequences of states are constructed. These sequences, also called traces (and interpreted as temporal models), formally describe behaviour. They adhere to the compositional structure: a trace describes a composed information state that changes in time (see Figure 8).

**Definition 3.3 (trace and temporal model)** A *trace* or *partial temporal model* of information type $\Sigma$ is a sequence of information states $(M^t)_{t \in \mathbf{N}}$ in $IS(\Sigma)$. The set of all partial temporal models is denoted by $IS(\Sigma)^{\mathbf{N}}$, or $\mathrm{Traces}(\Sigma)$.

A set of partial temporal models is a declarative description of the semantics of the behaviour of the system; each temporal model can be seen as one of the alternatives for the intended behaviour.

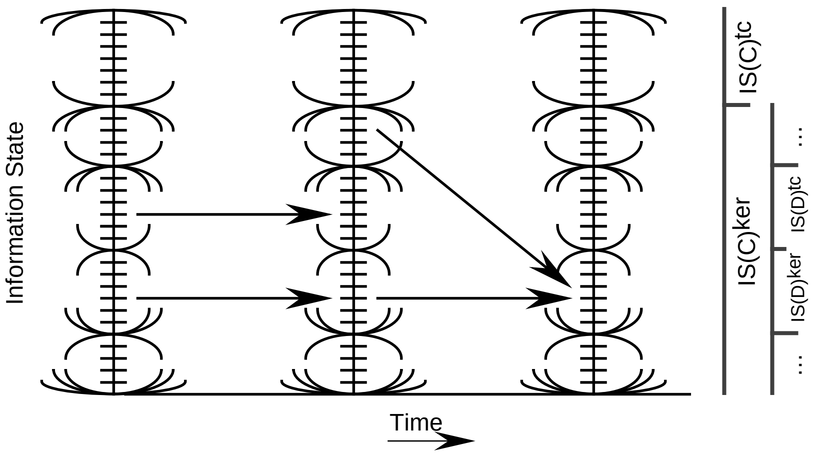


**Fig. 8** Compositional information states in time.

## 4 COMPOSITIONAL INFORMATION STATES

The compositional structure of composed tasks is reflected in composed information states, as shown in Figure 8. The information state of a component changes as a result of (1) input received from other component, or (2) the execution of the corresponding task.

The execution of the corresponding task changes both the internal information state of the component (private information) and the output information made available to other components (public information). The private information is based on internal information types and information types in the public input and output interface; during execution inputs are taken from the input interface and outputs are transferred to the output interface. The distinction between public and private information within a component is shown in Figure 6. Also the distinction between task (control) related information and kernel related information is shown in Figure 6.

In this section, components may be either composed or primitive, unless explicitly specified.

**Definition 4.1 (public kernel information)** Each component $C$ is assigned an information type, called the *public kernel information type* of $C$, denoted by $\Sigma_{C,pub}{}^{ker}$. The *input* and *output parts* are sub-information types, denoted by $\Sigma_{C,in}{}^{ker}$ and $\Sigma_{C,out}{}^{ker}$. The *public kernel information state* for $C$ is the information state for information type $\Sigma_{C,pub}{}^{ker}$ and the fixed *set of public kernel information states* for $C$ is denoted by $IS_{pub}{}^{ker}(C)$.

Kernel information states can be either primitive or composed. The atoms that define the basis of a primitive kernel information state are those specified within the kernel of a primitive component (e.g., the information type of a knowledge base). The kernel information state of a composed component is a combination of the kernel information states of the components within the composed component.

**Definition 4.2 (kernel information state)** Let $D$ be a component.

a) If $D$ is primitive, then its *set of private kernel information states* is defined by:

$$IS_{priv}{}^{ker}(D) = IS(\Sigma_{D,priv}{}^{ker})$$

In this definition $\Sigma_{D,priv}{}^{ker}$ is the overall private information type assigned to $D$: it combines both internal elements and copies of information types of public (input and output) interface.

b) If $D$ is composed and its *set of children components* is denoted by $Ch(D)$, the *set of private kernel information states* for $D$ is recursively defined by:

$$IS_{priv}{}^{ker}(D) = \prod_{C \in Ch(D)} IS^{ker}(C)$$

c) For any component $D$ the *set of kernel information states* is the combination of public and private information states

$$IS^{ker}(D) = IS_{priv}{}^{ker}(D) \times IS_{pub}{}^{ker}(D)$$

In this combination there is still a distinction between public information states (see definition 4.1) and private information states.

An example of a composed information state is shown in Figure 12, where each of the columns depicts a composed information state.

The private part of the information state of the *task control* of a component can be constructed in a similar manner. Task control knowledge is used to supervise the activation of the kernel (both sub-components and information links). Task control information includes specification of sets of goals and requests, exhaustiveness of search, etc: the additional information required to execute a task.

**Definition 4.3 (task control information state)** The task control of a component D is modelled as a primitive component. Similar to definitions 4.1 and 4.2 an information state is defined.

a) For any component D the *set of public task control information states* is defined by:

$$\mathrm{IS}_{\mathrm{pub}}{}^{\mathrm{tc}}(\mathrm{D}) \;=\; \mathrm{IS}(\Sigma_{\mathrm{D,pub}}{}^{\mathrm{tc}})$$

that, in turn, is split into input and output information state.

b) For any component D the *set of private task control information states* is defined by:

$$\mathrm{IS}_{\mathrm{priv}}{}^{\mathrm{tc}}(\mathrm{D}) \;=\; \mathrm{IS}(\Sigma_{\mathrm{D,priv}}{}^{\mathrm{tc}})$$

c) For any information link I the *set of (public) task control information states* is denoted by: $\mathrm{IS}^{\mathrm{tc}}(\mathrm{I})$

d) The *task control information* state of a primitive component C is defined as

$$\mathrm{IS}^{\mathrm{tc}}(\mathrm{C}) = \mathrm{IS}_{\mathrm{pub}}{}^{\mathrm{tc}}(\mathrm{C}) \times \mathrm{IS}_{\mathrm{priv}}{}^{\mathrm{tc}}(\mathrm{C})$$

e) If a component D is composed, the *set of composed task control information states* of the component D is defined as the composition of all task control information of children components C and kernel links I of D (formally defined in Section 5.1):

$$\mathrm{IS}^{\mathrm{com,tc}}(\mathrm{D}) = \prod_{\mathrm{C} \in \mathrm{C(D)}} \mathrm{IS}^{\mathrm{tc}}(\mathrm{C}) \times \prod_{\mathrm{I} \in \mathrm{KL(D)}} \mathrm{IS}^{\mathrm{tc}}(\mathrm{I})$$

f) The *task control information* state of a composed component D is defined as

$$\mathrm{IS}^{\mathrm{tc}}(\mathrm{D}) = \mathrm{IS}_{\mathrm{pub}}{}^{\mathrm{tc}}(\mathrm{D}) \times \mathrm{IS}_{\mathrm{priv}}{}^{\mathrm{tc}}(\mathrm{D}) \times \mathrm{IS}^{\mathrm{com,tc}}(\mathrm{D})$$

Definitions d) and f) combine all parts of a component that contain control information and at the same time distinguish the interface (public) control information from private control and possibly the control of each subcomponent.

Knowledge about the private task control is specified explicitly in DESIRE; the information types used are generic (including relations such as previous_component_state, previous_link_state, next_component_state, next_link_state, ...), see Section 5.2.

The kernel and the task control together define the information state of components.

**Definition 4.4 (component information states)** Let D be any component.
The *set of information states* of D is defined as:

$$IS(D) = IS^{ker}(D) \times IS^{tc}(D)$$

This definition reflects the distinction between kernel and control information defined in every component. Note that a component information state defines a rich information structure: it includes information states for all of its sub-components. For a given component information state, to focus on the state of a sub-component, a projection can be made on this sub-component, which leaves out all information of other states. In contrast, to abstract from the sub-components and their states, they simply can be left out, which leaves an information state at the abstraction level of the component.

In Figure 9 a component is shown with a trivial composition. The *information state* of component C is depicted in the component D where the possible sets of information states are identified.

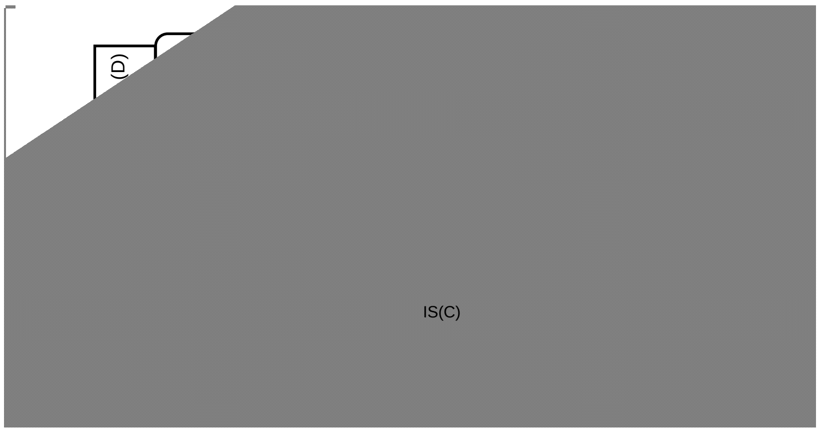


**Fig. 9.** Pictorial representation of an information state in a component.

Another distinction in the structure of the information state can be made. The kernel input and output interface are not merely a collection of information types, but are viewed as a *combination* of information types, where information types are grouped in levels, and levels are ordered in an object/meta/meta-meta/…-relation. In the context of compositional structures and for the specification of detailed process information the notion of levelled information types is important. The specification of more detailed dynamics is modelled through (meta-level) reasoning of one level about the state of the process of the level below.

## 5 COMPOSITIONAL BEHAVIOUR DESCRIPTIONS

The notion of a transition relation can be generalised for the compositional case, defining a compositional transition relation. A binary relation is defined between a tuple of information states (left hand side, precondition of the transition) and another tuple of information states (right hand side, result of the transition). This transition relation between one part of a compositional state and another part of a compositional state induces a transition relation for the compositional state as a whole.

Transition relations exist both within and between components. An information state of a component can have changed either because a component has been active and generated new information itself, or because information has been transferred/exchanged from one component to another. Transitions between components are specified by information links as defined below. Transitions within components are either transitions within the kernel, transitions within the task control, or transitions between task control and the kernel. Traces are generated as a result of such transitions. These traces can be interpreted as the behaviour of the system.

### 5.1 Transitions between components

To enable information exchange between components, information links are specified: between two components (private links) or between (the interfaces of) a component and one of its sub-components (a mediating link), as shown in Figure 10. These mediating links provide the means to connect two different levels of abstraction: the component and its components.

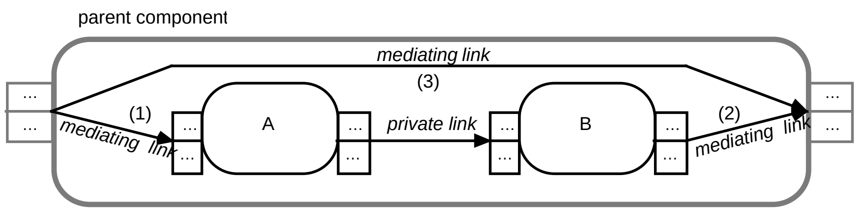


**Fig. 10** Kernel information links.

Information links are the basis for information exchange between components. The interface of every component consists of one or more (meta-)levels as explained above. Activation of a link from component $C_1$ to component $C_2$ causes a change in the information state of $C_2$ on the basis of information available in $C_1$. This change is a refinement or (for updates) a non-conservative modification of the information state of $C_2$. In effect it implies an extension, update, or revision of the information state. A (relative) principle of conservation is assumed: all information that is not explicitly changed by an activation of a link will remain available (a specific frame assumption). The task control component controls the activation of the various links.

An information link is defined on the basis of semantic units: atoms and their truth values. It relates a semantic unit of a component $C_1$, defined by a pair $< a, tv_1 >$ of a ground atom $a$ of one information type and a truth value $tv_1$ to a semantic unit of another component $C_2$, defined by a pair $< b, tv_2>$ with $b$ of another information type. Atoms which refer to the same entities in the world may be named differently within different components, in which case an information link defines the "translation" of atom names.

**Definition 5.1 (semantic unit, kernel information link)**

Let $D$ be a complex component and $C$, $C_1$ and $C_2$ its components. The set of *public semantic units* of level $x$ for the kernel of a component $C$ is defined by

$$SU^{(x)}(C) = At(\Sigma_{C,pub}^{ker,(x)}) \times \{0, 1, u\}.$$

$SU^{(x)}_{in}(C)$ resp. $SU^{(x)}_{out}(C)$ denotes the restriction of this set to the input resp. output of component C.

a) A *private kernel link* of $D$, $l$ *from level* $x$ *of* $C_1$ *to level* $y$ of $C_2$ is a binary relation

$$l : SU^{(x)}_{out}(C_1) \times SU^{(y)}_{in}(C_2)$$

b) Likewise, the *mediating kernel links* are defined as

$$l : SU^{(x)}_{in}(D) \times SU^{(y)}_{in}(C)$$

$$l : SU^{(x)}_{out}(C) \times SU^{(y)}_{out}(D)$$

$$l : SU^{(x)}_{in}(D) \times SU^{(y)}_{out}(D)$$

An example of a mediating link of the first type listed in Definition 5.1.b) is link 1 in Figure 10, an example of the second type is link 2, and of the third type link 3. A trivial standard example of an information link is when the sets of semantic units have a common subset $U$, and for $< a, tv_1 > \in SU^{(x)}_{out}(C_1)$, $< b, tv_2 > \in SU^{(y)}_{in}(C_2)$ the identity relation $I$ is defined by

$I(\langle a, tv_1 \rangle, \langle b, tv_2 \rangle)$ if $\langle a, tv_1 \rangle = \langle b, tv_2 \rangle \in U$. The amount of information transferred is regulated by the set $U$, this set can be defined to its maximum size (transferring all possible semantic units), reduced to empty (transferring no semantic units at all), or defined as any size in between these extremes (transferring a specific subset of semantic units).

In the example below, the name of the parameter to be deduced is passed to the component that deduces an additional value. The transfer specified by a private link translates poss_ass( likely_candidate( X: RequirementParameter ) ) to the meta-statement assumption(candidate(X: RequirementParameter), pos) which translates within the input interface into the object level statement candidate(X: RequirementParameter).

```
private link pass_to_deduce_reqparm: object-assumption
  domain make_assumptions_on_candidate_parameter
    output output_1
  co-domain determine_parameter_suitable_for_extension
    input input_2
  sort links
    ( RequirementParameter, RequirementParameter )
    ( Value, Value )
  object links identity
  term links identity
  atom links
    ( poss_ass( likely_candidate(P: RequirementParameter ) ),
      assumption( candidate( P: RequirementParameter ), positive ) ):
      < <true, true>, <false, false>, <unknown, false> >;
end link
```

The dynamic semantics of such information links can be expressed by the notion of transition introduced in Section 3. Note that a simple transition between one or more factors of a cartesian product can be extended in a canonical manner to a relation on the whole cartesian product. Thus any transition relation

$\pi : (A \times B) \times (B \times C)$ with $\pi(\langle a, b \rangle, \langle b', c' \rangle)$

*induces* the relation $\pi^* : (A \times B \times C) \times (A \times B \times C)$

that is $\pi^*(\langle a, b, c \rangle, \langle a', b', c' \rangle)$ with $a' = a$

whenever $\pi(\langle a, b \rangle, \langle b', c' \rangle)$

Factors that are not influenced by the transition can be added on the left hand side or left unchanged in the transition (conservation).

**Definition 5.2 (information link transition)** Suppose $E_1$ and $E_2$ are components and $I$ is an information link from level $x$ of $E_1$ to level $y$ of $E_2$.

A *transition relation for the information link* $I$ is a relation

$\pi : (IS_x(E_1) \times IS_y(E_2) \times IS^{tc}(I)) \; \times \; (IS_y(E_2) \times IS^{tc}(I))$

such that for all $M_1, M_2, N, M'_2, N'$ with $\pi(< M_1, M_2 , N >, < M'_2, N' >)$

and for any atom $b$

$M'_2(b) = M_2(b)$ (conserved) or

$I(< a, M_1(a) >, < b, M'_2(b)>)$ for some atom $a$ (changed by the link)

## 5.2. Transitions due to task control

Task control is specified explicitly in DESIRE; the information types used are generic. In the example below the task control of the component requirement extension determination is specified: this task can be activated to determine a parameter (indicated by the evaluation criterion in the condition), terminating when a default value has been determined.

```
task control requirement_extension_determination
  task control knowledge
    .......
    if previous_component_state( extension_suitability_determination, active)
      and component_state( extension_suitability_determination, idle )
      and evaluation( extension_suitability_determination, all_output, any, succeeded )
      then next_component_state( extension_method_determination, active)
      and next_link_state( epistemic_info_on_suitable_for_extension, uptodate );
    .......
    if previous_component_state( default_requirement_determination, active )
      and component_state( default_requirement_determination, idle )
      and evaluation( default_requirement_determination, assign_default, any, succeeded )
      then stop
      and next_link_state( default_extended_parameter, uptodate );
    ........
end task control
```

**Definition 5.3 (task control transition)** A *task control transition relation for a component* $C$ is a relation associating task control information states for $C$ to task control information states for $C$; i.e., a relation $\pi : IS^{tc}(C) \; \times \; IS^{tc}(C)$ where each transition is induced by the task control specification.

The task control information must, in some way, be transferred between the task control of the parent component and the task control information of the sub-components. Three kinds of task control links are discerned: upward task control links, and downward task control links that connect the private task control to the sub-component and sub-link task control, and thirdly the mediating task control links that communicate the private task control to the public task control and vice versa (see Figure 11).

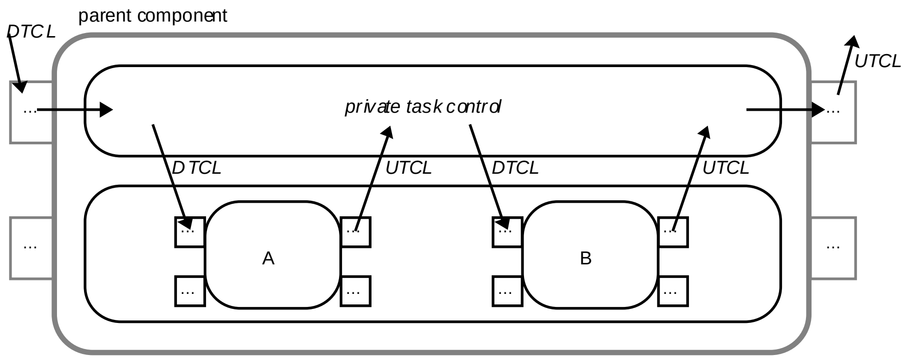


**Fig. 11** Task control links

Formally, task control links are defined as follows. For example, each component C has a downward link defined by the pair < next-component-state(C, active), 1 >, < component-state(active), 1 >.

**Definition 5.4 (task control link)** Let D be a composed component and let the set of semantic units $SU_{priv}^{tc}$ be defined by $SU_{priv}^{tc}(D) = At(\Sigma_{D,priv}^{tc}) \times \{0, 1, u\}$.

a) A (combined) *downward task control link* (denoted by DTCL) is a set consisting, for each component C, of a relation on

$l : SU_{priv}^{tc}(D) \ \times \ SU_{in}^{tc}(C)$

where $SU_{in}^{tc}$ is defined by $SU_{in}^{tc}(C) = At(\Sigma_{C,pub}^{tc}) \times \{0, 1, u\}$

b) A (combined) *upward task control link* (denoted by UTCL) is a set consisting, for each component C, of a relation on

$l : SU_{out}^{tc}(C) \ \times \ SU_{priv}^{tc}(D)$

where $SU_{out}^{tc}$ is defined by $SU_{out}^{tc}(C) = At(\Sigma_{C,pub}^{tc}) \times \{0, 1, u\}$

c) A (combined) *mediating task control link* (denoted by MTCL)is a set of relations

$l : SU_{in}^{tc}(D) \ \times \ SU_{priv}^{tc}(D)$ or $l : SU_{priv}^{tc}(D) \ \times \ SU_{out}^{tc}(D)$

Note that in the specification language within DESIRE these task control links are implicit. In contrast to kernel links, it is not necessary to specify these control links.

**Definition 5.5 (task control link transition)** Let D be a composed component.

a) A *transition relation for the upward task control link* UTCL is defined as a relation

$\pi_{UTCL} : (IS^{com,tc}(D) \times IS^{tc}(D)) \times IS^{tc}(D)$

such that for all $N_1, N_2, N'_2$ in $\pi_{UTCL}(<N_1, N_2>, N'_2)$ it holds that for any atom b :

$N'_2(b) = N_2(b)$ (conserved) or

$I(< a, N_1(a) >, < b, N'_2(b)>)$ for some atom a (changed by a link).

b) A *transition relation for the downward task control link* DTCL is defined as a relation

$\pi_{DTCL} : (IS^{tc}(D) \times IS^{com,tc}(D)) \times IS^{com,tc}(D)$

such that for all $N_1, N_2, N'_2$ in $\pi_{DTCL}( <N_1, N_2>, N'_2)$ it holds that for any atom b :

$N'_2(b) = N_2(b)$ (conserved) or

$I(< a, N_1(a) >, < b, N'_2(b)>)$ for some atom a (changed by a link).

## 5.3. Compositional Transitions and Traces

For primitive reasoning components, kernel transitions are induced by inferences on the basis of knowledge in the knowledge base. Compositional transitions define the dynamic semantics of hierarchical compositional systems.

**Definition 5.6 (kernel transitions)** Let C be a component. A *kernel transition relation for a component* C (or *private component transition relation*) is a relation associating information states for C to information states for C; i.e., a relation

$\pi : IS^{ker}(C) \times IS^{ker}(C)$

where each transition is induced by either a kernel transition relation of a (child) component or by a transition of an information link.

**Definition 5.7 (compositional transitions)** Let D be a composed component with sub-component C. A *compositional transition relation* for the D is a transition relation

$\pi : IS_{priv}(D) \times IS_{priv}(D)$

where each transition is induced by a transition of one of the following types:

(1) a kernel transition,

(2) a task control link transition,

(3) a task control transition.

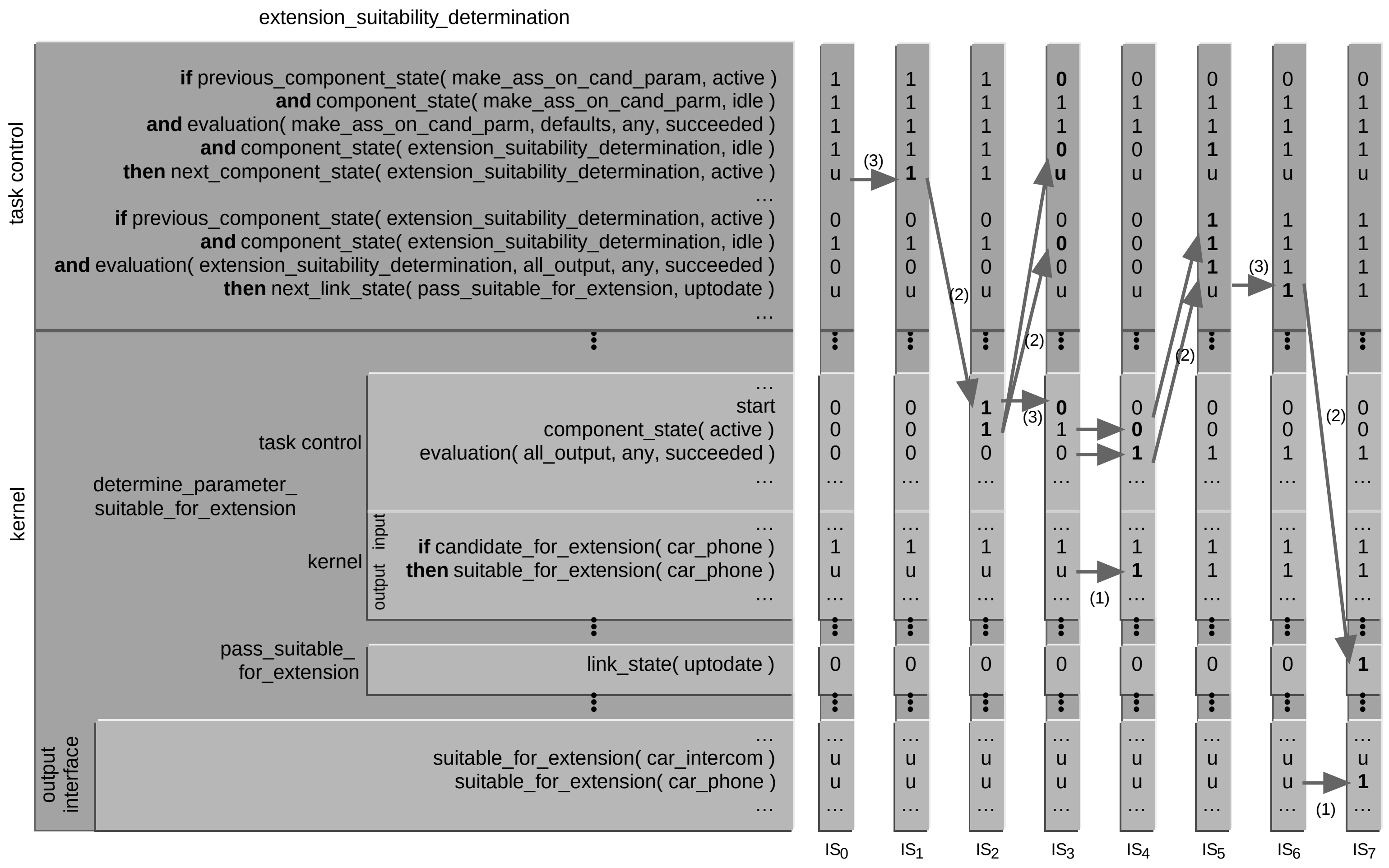
In Figure 12 the three subtypes of compositional transitions are shown. The information states change according to the specifications given of kernel contents (above), task control contents (Section 5.2), and the implicit task control links.

All information states depicted in Figure 12 are information states of the component extension suitability determination (see Figures 3, 4, and 5 for more details). In the original information state $IS_0$ the task control of the parent component has information that the parent is currently ‘starting’, and that inspection of the state of its components revealed that the component extension suitability determination is currently idle. Note that in this trace of information states per default conservatism holds: partial truth values are only changed by means of explicit transitions.

In Figure 12 the sets of atoms depicted on the left have a structure that is identical to the task composition. Note that several knowledge rules are specified: this is for the purpose of explaining the trace. In the trace changes to an information state with respect to the previous information state are emphasized by a **bold** typeface. The transitions to each information state are described below.

- The transition to the next information state $IS_1$ is induced by a (3) task control transition. The task control of the parent component has inferred that in the next information state the state of the component extension suitability determination should become active.
- The transition to $IS_2$ is induced by a (2) task control link transition (downward) in which the start atom of the component extension suitability determination is made true. At the same time, the state of this component is changed from idle to active.
- The transition to $IS_3$ is induced by a (2) task control link transition (upward), in which the current state of the component is transferred to the parent component (which also then revises its previous conclusions, also taking into account the step in time; e.g., what was current becomes previous). Note that when a component is active, it automatically is no longer idle.
- The transition to $IS_4$ is induced by a (1) kernel transition: by the knowledge base of the component determine parameter suitable for extension it has been inferred that the atom suitable_for_extension( car_phone ) holds. This also causes the success of the evaluation criterion all_output to hold (i.e. become true) under the extent of any.

- The transition to $IS_5$ is induced by a (2) task control transition (upward), in which the current state of the component is transferred to the parent component. In the task control of the parent component it is now known that previously the component extension suitability determination was active, currently it is idle, and a particular evaluation criterion of that component has succeeded.
- The transition to $IS_6$ is induced by a (3) task control transition. The task control of the parent component has inferred that in the next information state the link pass suitable for extension needs to become uptodate. This is then illustrated in the transition to the next information state.
- The transition to $IS_7$ is induced by two transitions. An information link has become uptodate via a (2) task control transition (downard). (Partial) truth-values are transferred from their source to their destination via a (1) kernel transition. This implies that, in the output interface of the parent component, now also the atom suitable_for_extension( car_phone ) holds.



extension_suitability_determination
task control
if previous_component_state( make_ass_on_cand_param, active )
and component_state( make_ass_on_cand_parm, idle )
and evaluation( make_ass_on_cand_parm, defaults, any, succeeded )
and component_state( extension_suitability_determination, idle )
then next_component_state( extension_suitability_determination, active )
if previous_component_state( extension_suitability_determination, active )
and component_state( extension_suitability_determination, idle )
and evaluation( extension_suitability_determination, all_output, any, succeeded )
then next_link_state( pass_suitable_for_extension, uptodate )
kernel
determine_parameter_ suitable_for_extension
task control
start
component_state( active )
evaluation( all_output, any, succeeded )
kernel
input
output
if candidate_for_extension( car_phone )
then suitable_for_extension( car_phone )
pass_suitable_ for_extension
link_state( uptodate )
output interface
suitable_for_extension( car_intercom )
suitable_for_extension( car_phone )
(1)
(2)
(3)
IS0
IS1
IS2
IS3
IS4
IS5
IS6
IS7

**Fig. 12** A trace of information states.

The following definition shows how traces generated by iteratively applying a transition function on the current information state can be interpreted as temporal models. These temporal models provide a declarative description of the semantics of the behaviour of the system; they can be viewed as the intended (behavioural) models of the system.

**Definition 5.8 (compositional trace)** Let $D$ be a component.

a) A *trace* of a component $D$ is a sequence of information states $(M^t)_{t \in \mathbf{N}}$ in $IS^{com}(D)$. The set of all traces is denoted by $IS(D)^{\mathbf{N}}$, or $Traces(D)$.

b) An element $(M^t)_{t \in \mathbf{N}} \in Traces(D)$ is called *a temporal model of* $D$ if for all time points $t$ the step from $M^t$ to $M^{t+1}$ is defined in accordance with a compositional transition of the system. The set of temporal models of $D$ forms a subset $BehMod(D)$ of $Traces(D)$.

A temporal model describes a trace representing possible (intended) behaviour of the reasoning. One view is that the trace is generated by the (execution of) transition functions, given initial input information. From every initial information setting a trace can be generated by the transitions. Together the generated traces form the set $BehMod(D)$. A slightly different view is that the transition relations define a set of (temporal) axioms or constraints $BehTheory(D)$ on temporal models in $Traces(D)$. The possible behavioural alternatives are given by the set of the temporal models satisfying these temporal constraints:

$$TempMod(D) = \{ M \in Traces(D) \mid M \models BehTheory(D) \}$$

where $M \models BehTheory(D)$ holds iff each formula from the set $BehTheory(D)$ has truth value *true* at every time point in the temporal model $M$. This second view provides a formalisation of the intended behavioural patterns in the form of the (intended) models of a logical (temporal) theory in a specific type of temporal logic, giving a declarative (Tarski) semantics. The formal semantics of the behaviour is defined by the set of models $TempMod(D)$. The first view corresponds to the notion of an executable temporal logic. Both views co-exist: executing a temporal theory is a useful technique to construct a model of this theory.

In a compositional trace, information states for components at all levels of abstraction of a compositional system are included. To abstract from the lower levels of abstraction, all information states of sub-components can be left out of the states of the trace. The remaining trace shows the behaviour at the abstraction level of the system as a whole. For transitions that are hidden from this high abstraction level, stuttering steps will be found in the abstract trace.

## 5.4. Temporal semantics of task control

The temporal semantics of the task control can now easily be defined. The behaviour of a system (i.e., the tasks in the kernel of a component) results in a trace of information states. The predicates (defined in a generic manner) in the task control of the encompassing component refer to two of such information states: the *current* information state and the *previous* information state. All conclusions drawn by the task control refer to the *next* information state, i.e. what is to happen.

Figure 13 depicts this for an example task control rule. As time passes, different information states are produced. The *current information state* is always the latest information state that is produced. This then automatically defines the previous information state.

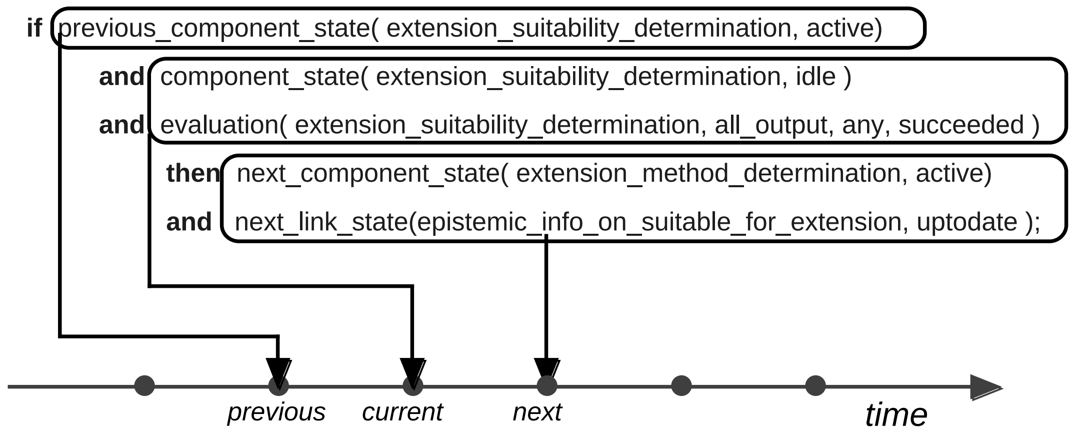

if previous_component_state( extension_suitability_determination, active)
and component_state( extension_suitability_determination, idle )
and evaluation( extension_suitability_determination, all_output, any, succeeded )
then next_component_state( extension_method_determination, active)
and next_link_state(epistemic_info_on_suitable_for_extension, uptodate );
previous
current
next
time

**Fig. 13.** Temporal semantics of an example task control rule.

The statements derived about the next information state restrict which component and information link will become active with particular control settings. Of course no "predictions" are made about the success or failure of a task in the next information state.

On the operational side, it is not too difficult to make sure that whatever is derived in the task control about the next state, happens in the next information state. This facilitates the operationalisation of an entire task model, and thus enables automatic generation of prototype implementations.

## 6 DISCUSSION AND CONCLUSIONS

The compositional development method DESIRE is based on the assumption that dynamic aspects are essential for modelling complex tasks and processes. The temporal semantics approach to the description of a compositional system's behaviour presented provides a means to describe the dynamics of compositional task models and problem solving methods. The state of a composed component, at any given point in time, is described as a combination of the states of the composed component's sub-components and the state of the task control. The compositional structure of information states, transitions and reasoning traces provides a transparant model of the system's behaviour, both conceptually and formally. Such a model of a system's behaviour serves as the basis for temporal reasoning on the control of the behaviour of the system.

Another knowledge engineering method that supports hierarchical task structures is MIKE/KARL [3], [4] in which inference actions can be defined in terms of other (more primitive) inference actions. This language is related to the KADS methodology [33] with its three independent (hierarchically organised) layers. In KADS, hierarchical composition is only available at the task layer. The introduction of hierarchical composition at the inference layer in KARL is an extension of KADS. In KARL the composition of the inference layer is in a one to one correspondence to the composition of the task layer. Similar to the approach here the semantics attributed to a composed inference action takes into account the related control structure at the task layer and the primitive inference actions included. A difference is that in our specification this related control structure is included in the composed component itself and not separated at a distinct (task) layer. This implies that in our case the specification document mirrors the compositional structure more explicitly, in the sense of 'hiding' the control inside the composed component.

Moreover, in KARL the temporal aspects are not explicitly covered by the formal semantics that assigns a 'static' (input-output) semantics, based on dynamic logic [17]. In particular, for applications in which the interaction between components that reason and act autonomously, such as co-operative (human-computer) systems and multi-agent systems, the dynamics of interactions is crucial. These interactions take place in a dynamically controlled manner and can be modelled in our temporal framework by transitions between states. As KADS is based on a different conceptual model, in which the notions of state and transition are left implicit, KADS languages like KARL and (ML)$^2$ have difficulty expressing dynamically controlled interactions between autonomous components in an adequate manner. In this respect TASK [28] allows for more dynamics and therefore is closer to DESIRE.

A formal basis to the conceptual phase of complex system design, supporting knowledge acquisition and within which behaviour can be explicitly modelled, is presented in this document. Reuse of formally specified components of a system is possible. The behaviour of the components in interaction with other components can be well-defined. In current research multi-agent situations, in which agents are modelled as interacting components, are being explored; e.g., [7].

Given compositional descriptions of complex dynamic systems, together with well-defined semantics, validation and verification of system behaviour should be possible. Initial research in this area is promising [15], [23] .

## ACKNOWLEDGEMENTS

This research was partly funded by the Netherlands Organization for Scientific Research (NWO) within the REVISE project “Evolutionary design in knowledge-based systems” (project number 612-322-316) and the ESPRIT III Basic Research project 6156 DRUMS II.

## REFERENCES

1. Agusti, J., Esteva, F., Carcia, P., Godo, L., Lopez de Mantaras, R., Murgui, Ll., Puyol, J., and Sierra, C. (1992). Structured Local Fuzzy Logics in MILORD, In: Zadeh, L., and Kacprzyk, J. (eds.) *Fuzzy Logic for the Management of Uncertainty,* John Wiley & Sons, Inc.

2. Akkermans, J.M., Harmelen, F. van, Schreiber, A.Th., and Wielinga, B.J. (1992). A formalisation of knowledge-level models for knowledge acquisition. *Int. J. Of Intelligent Systems*.

3. Angele, J., Decker, S., Perkuhn, R. and Studer, R. (1996). Modelling Problem-Solving Methods in New KARL. In: Gaines, B.R., and Musen, M.A. (eds.), Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-based Systems workshop (KAW'96), Calgary: SRDG Publications, Department of Computer Science, University of Calgary, pages 1/1-1/18.

4. Angele, J., Fensel, D., and Studer, D. (1996). Domain and Task Modelling in MIKE. Proceedings of the IFIP WG 8.1/13.2 Joint Working Conference, Domain Knowledge for Interactive System Design. Geneva, Switzerland, May 8-10th, 1996.

5. Balder, J., Akkermans, H. (1990). *StrucTool: Supporting Formal Specifications of Knowledge-level Models*. In: [39], pp. 60-77

6. Blamey, S. (1986). Partial Logic. In: D. Gabbay and F. Günthner (Eds.), *Handbook of Philosophical Logic*. Vol. III, pp. 1-70, Reidel, Dordrecht.

7. Brazier, F.M.T., Dunin-Keplicz, B., Jennings, N., and Treur, J. (1995). Formal Specification of Multi-Agent Systems: a Real-World Case, *Proceedings First International Conference on Multi-Agent Systems, ICMAS'95,* pp. 25-32. An extended version appeared as: DESIRE: modelling multi-agent systems in a compositional formal framework, in: International Journal of Cooperative Information Systems, vol. 6 (no. 1), M. Huhns, M. Singh, (eds.), special issue on Formal Methods in Cooperative Information Systems: Multi-Agent Systems, 1997, pp. 67-94.

8. Brazier, F.M.T., Langen, P.H.G. van, Treur, J., Wijngaards, N.J.E. and Willems, M. (1996). DESIRE: Designing an elevator configuration. In: Schreiber, A.Th., and Birmingham, W.P. (eds.), Special Issue on Sisyphus-VT. *International Journal of Human-Computer Studies*, 1996, Volume 44, pp. 469-520.

9. Brazier, F.M.T., Treur, J. and Wijngaards, N.J.E. (1996). The acquisition of a shared task model. In: Shadbolt, N., O'Hara, K., and Schreiber, A.Th. (eds.). *Advances in Knowledge Acquisition; 9th European Knowledge Acquisition Workshop, EKAW'96,* Lecture Notes in Artificial Intelligence, Volume 1076, Springer Verlag, pp. 278-289.

10. Brazier, F.M.T., and Wijngaards, N.J.E. (1997). *A Purpose Driven Method for the Comparison of Modelling Frameworks*. In: [29], pp. 323-328.

11. Brazier, F.M.T., Treur, J., Wijngaards, N.J.E., and Willems, M. (1995). Formal specification of hierarchically (de)composed tasks. In Gaines, B.R. and Musen, M.A. (eds.), *Proceedings of the 9th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW '95,* Volume 2, pp. 25/1-25/20. Calgary: SRDG Publications, Department of Computer Science, University of Calgary.

12. Engelfriet, J. and Treur, J. (1994). Temporal theories of reasoning. In: MacNish, C., Pearce, D., and Pereira, L.M. (Eds.), *Logics in Artificial Intelligence, Proceedings of the 4th European Workshop on Logics in Artificial Inteligence, JELIA'94,* Springer-Verlag, volume 838 of Lecture Notes in Artificial Intelligence, pages 279-299.

13. Cesta, A. and Oddi, A. (1996). A Representation Language for Domain Knowledge in Planning Architectures. In: Gaines, B.R. and Musen, M.A. (Eds.), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW'96,* Volume 1, pp. 16/1-16/15, Calgary. SRDG Publications, Department of Computer Science, University of Calgary.

14. Chien, S.A. (1996). Knowledge Acquisition, Validation, and Maintenance in a Planning System for Automated Image Processing. In: Gaines, B.R. and Musen, M.A. (Eds.), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-Based Systems Workshop, KAW'96,* Volume 1, pp. 17/1-17/19, Calgary. SRDG Publications, Department of Computer Science, University of Calgary.

15. Cornelissen, F., Jonker, C.M., and Treur, J. (1997). *Compositional verification of knowledge-based systems: a case study in diagnostic reasoning*. In: [29], pp. 65-80.

16. Fensel, D. (1995) *The knowledge acquisition and representation language KARL*, PhD. Thesis, Univ. of Karlsruhe. Kluwer Academic Publisher, Boston, 1995.

17. Fensel, D., Harmelen, F. van, (1994) A comparison of languages which operationalize and formalize KADS models of expertise. *Knowledge Engineering Review*, Volume 9, pp. 105-146.

18. Ford, K.M., Bradshaw, J.M., Adams-Webber, J.R., and Agnew, N.M. (1993). Knowledge Acquisition as a Constructive Modeling Activity. In: K.M. Ford and J.M. Bradshaw (Eds.), *Knowledge Acquisition as Modeling, International Journal of Intelligent Systems*: Wiley and Sons, 1993, Vol. 8, Nr. 1, pp. 9-32.

19. Harmelen F. van, and Balder, J. R. (1992). $(ML)^2$: a formal language for KADS-models of expertise, *Knowledge Acquisition Journal*, vol. 4 (no. 1). Special issue: The KADS approach to knowledge engineering.

20. Gavrila, I.S. and Treur, J. (1994). A formal model for the dynamics of compositional reasoning systems, in Cohn, A.G. (Ed.), *Proc. 11th European Conference on Artificial Intelligence, ECAI'94* , John Wiley & Sons, Chichester, pp. 307-311.

21. Geelen, P.A., and Kowalczyk, W. (1992). A knowledge-based system for routing of international blanc payment orders, In: *Proc. of Int. Conf. on AI, Expert Systems and Natural Language, Avignon-92*, vol 2, pp. 669-677.

22. Harmelen, F. van, and Fensel, D. (1995). Formal methods in knowledge engineering. *The Knowledge Engineering Review*, Volume 10(4), pp. 345-360.

23. Jonker, C.M., and Treur, J., *Compositional Verification of Multi-Agent Systems: a Formal Analysis of Pro-activeness and Reactiveness*. In: [32], pp. 24.

24. Kowalczyk, W., Treur, J. (1990). *On the use of a Formalized Generic Task Model in Knowledge Acquisition*, In: [39], pp. 198-221.

25. Langevelde, I.A. van, Philipsen, A.W. and Treur, J. (1992). Formal specification of compositional architectures. In: Neumann, B. (Ed.), *Proc. 10th European Conference on Artificial Intelligence, ECAI'92*, John Wiley & Sons, Chichester, pp.

272-276. Extended version: *Report IR-282*, Vrije Universiteit Amsterdam, Department of Mathematics and Computer Science, 1991

26. Langen, P.H.G. van, and Treur, J. (1989). Representing World Situations and Information States by Many-Sorted Partial Models. *Technical Report PE8904, University of Amsterdam*, Department of Mathematics and Computer Science.

27. Langholm, T. (1988). Partiality, Truth and Persistence. *CSLI Lecture Notes, No. 15*. Stanford University, Stanford.

28. Pierret-Golbreich, C., and Talon, X. (1997). Specification of Flexible Knowledge-Based Systems. In: [29], pp. 190-204.

29. Plaza, E., Benjamins, R. (eds.) (1997). *Knowledge Acquisition, Modelling and Management, Proceedings of the 10th European Knowledge Acquisition Workshop, EKAW'97*, Lecture Notes in AI, vol. 1319, Springer Verlag, Berlin, 1997

30. Puerta, A.R., Egar, J.W., Tu, S.W., and Musen, M.A. (1992). A multiple-method knowledge-acquisition shell for the automatic generation of knowledge-acquisition tools. *Knowledge Acquisition*, 1992, vol. 4, pp. 171-196.

31. REVISE (project) (1996). A Purpose Driven Method for Language Comparison. In: Shadbolt, N., O'Hara, K., and Schreiber, A.Th. (eds.) *Advances in Knowledge Acquisition. 9th European Knowledge Acquisition Workshop, EKAW'96.* Lecture Notes in Artificial Intelligence, Volume 1076, pages 66-81.

32. Roever, W.P. de, Langmaack, H., Pnueli, A. (eds.) (1998). *Proceedings of the International Workshop on Compositionality, COMPOS'97*, Springer Verlag. In press, 1998

33. Schreiber, A.Th., Wielinga, B.J. and Breuker, J.A. (eds.) (1993). *KADS: A Principled Approach to Knowledge-Based System Development*. Academic Press, London.

34. Shadbolt, N., Motta, E., and Rouge, A. (1993). Constructing Knowledge-Based Systems. *IEEE Software*, November 1993, pp. 34-38.

35. Shahar, Y., Miksch, S., and Johnson, P. (1996). A Task-Specific Ontology for Design and Execution of Time-Oriented Skeletal Plans. In: Gaines, B.R. and Musen, M.A. (eds.), *Proceedings of the 10th Banff Knowledge Acquisition for Knowledge-*

*Based Systems Workshop, KAW'96*, Volume 1, pp. 17/1-17/19, Calgary. SRDG Publications, Department of Computer Science, University of Calgary.

36. Treur, J. (1994), Temporal Semantics of Meta-level Architectures for the Control of Reasoning, In: Turini, F. (ed.), *Lecture Notes in Comp. Sc. 883*, Springer-Verlag.

37. Treur, J. and Wetter, Th. (eds.) (1993). *Formal Specification of Complex Reasoning Systems*, Ellis Horwood.

38. Wetter, Th. (1990). *First Order Logic Foundation of the KADS Conceptual Model*. In: [39], pp. 356-375

39. Wielinga, B.J., Boose, J., Gaines, B.R., Schreiber, A.Th., and Someren, M.W. van (eds.), *Current Trends in Knowledge Acquisition (Proc. EKAW'90)*, IOS Press, 1990