AUTHORING FOR INTELLIGENT SIMULATION BASED INSTRUCTION: A MODEL BASED APPROACH¹

TON DE JONG

*Eindhoven University of Technology
Department of Philosophy and Social Sciences
P.O. Box 513
NL-5600 MB Eindhoven
The Netherlands &
University of Amsterdam
Department of Social Science Informatics*

KENNETH TAIT

*University of Leeds
Computer Based Learning Unit*

WOUTER R. VAN JOOLINGEN

*Eindhoven University of Technology
Department of Philosophy and Social Sciences*

ABSTRACT. The general objective of the SIMULATE project (a Task Force within the DELTA SAFE Project) is to outline the requirements and (global) specifications of an authoring workbench that will support the creation of ISLEs (Intelligent Simulation Learning Environments). The project has made an inventory of potential elements of ISLEs and an inventory of tools for creating these elements. The present paper focuses on the project's work of 'formalisation': a description of elements of ISLEs in a structured ('formal') language placed somewhere between a description in a natural language and a description in a programming language. In addition to providing a language, formalisation identifies generic elements, relations and structures of ISLE-related knowledge. A future step will be the design of what we have called basic building blocks (BBBs) that form the 'software' counterparts of the structures. These BBBs can be tailored (by an author) to produce a specific ISLE through the processes of specialisation and instantiation. This view is compatible with the anticipated functional architecture of an ISLE, in which a generic ISLE shell is filled with an ISLE knowledge base that is created by an author. Also the notion of (reusable) building blocks, is very much in line with the OS-ID notion as followed in SAFE, and is related to the general DELTA approach in standardising aspects of authoring for open learning material, and, finally, it reflects modern opinions on model based development of Knowledge Based Systems.

¹This paper is based on work performed within the SAFE project, partially funded under contract number D1014 of the Exploratory action of the DELTA Programme. Guus Schreiber, Robert de Hoog, and Paul Kamsteeg (UvA) provided comments at an earlier version of this paper. We thank all our colleagues from the SIMULATE project for their contribution, as well as two anonymous reviewers for their helpful suggestions. We like to emphasise that the work reported in this paper is still under progress.

1. Introduction

The SIMULATE Task Force within the SAFE project addresses *learning and instruction with computer-based simulations* and, in particular, *authoring* for learning and instruction with such simulations. The starting point of the project is the observation that in order to be effective, simulations require the presence of an instructor or tutor to provide instructional support. For a computer-based simulation to be effective in an open-learning context in the absence of a human tutor it is necessary to embed the simulation in an adaptive learning environment which we term an Intelligent Simulation Learning Environment (ISLE). These ISLEs will be created with the help of an authoring tool which is called SIMULATE (SIMULAtion Authoring Tools Environment).

1.1. INTELLIGENT SIMULATION LEARNING ENVIRONMENTS

Modern learning theories state that in the study process the learner is an active agent who is an active constructor and reconstructor of his/her knowledge base. Some forms of Computer Assisted Instruction are well in line with this notion. The use of hypermedia systems, in which learners are encouraged to explore a domain, is such an example. Hypermedia based learning is currently topic of research in the SAFE task force HYPERATE. A second example of CAI that elicits exploratory behaviour is simulation based learning, the subject of the SAFE task force SIMULATE.

It is, however, also clear that exploratory learning puts a high cognitive demand on the learner. Instructional support is needed if learning from simulations is to be effective. This support can be provided by a computer environment as is shown by systems such as QUEST (troubleshooting of simple electronic circuits; White & Frederiksen, 1987), STEAMER (operating a steam propulsion plant, Hollan et al., 1984), Smithtown (micro-economics; Shute & Glaser, 1990), and others.

One of the first activities in the SIMULATE project has been to create an inventory of potential elements of ISLEs. This inventory took as its starting point four characteristics of instructional use of simulations: *Presence of formalized, manipulable underlying models*, *presence of learning goals*, *elicitation of specific learning processes* (such as hypotheses generation, predicting, and model exploration) and, finally, the *presence of learner activity* (meaning that the learner actually can *manipulate* something in the simulation).

In the inventory the consequences of these four characteristics on designing ISLEs, which are in fact ITSs, was assessed for each of the four 'classical' components of an ITS: the domain, the learner, the instruction and the interface component. The complete inventory can be found in de Jong (1990a).

One important distinction we made was the distinction between *directive* and *non-directive support*. *Directive support* (or guidance) steers the learner in a certain direction and in this way takes decisions from the learner and puts it in the hands of the system, e.g. when the learner gets direct feedback and/or hints for directions to follow (for example feedback such as "it is better to change only one variable at a time"). *Non-directive support* helps the learner with accomplishing what s/he would have done in a completely free exploratory environment (the best examples here are *dedicated scratchpads*: scratchpads that are meant to support a specific learning process, such as *hypotheses scratchpads* (see also de Jong 1990a; de Jong & Njoo, 1990)).

An Intelligent Simulation Learning Environment is conceived as being a learning

environment that offers the learner a view at the underlying model, helps in stating and monitoring learning goals, support learning processes (in a directive, non-directive way or both) and enables the realisation of learner activity.

1.2. SIMULATION AUTHORIZING TOOLS ENVIRONMENT

The aim of the SIMULATE project in this phase is to formulate the requirements and (global) specifications of an authoring tool that helps in creating ISLEs. It is therefore that the project has created a second inventory that listed all tools available for creating each of the four ITS components. For example, and most directly related to developing simulations, the project has made a catalogue of a large number of tools to create simulations (van Joolingen et al., 1990). The complete inventory is given in Tait (1990).

The SIMULATE authoring tool is intended to be an integrated workbench that consists of a number of components. The final SIMULATE system is envisaged to consist of a permanent *shell* that basically contains a set of interpreters, that is filled with an *ISLE knowledge base*. This knowledge base contains *default knowledge*, functionality of an ISLE that is always present, and *author generated knowledge* (see for a description Hijne & van Berkum, 1990). For providing this author generated knowledge the author is supported by a set of software tools in the form of library of basic building blocks of ISLEs, by a set of rules that constitute *recommendations* of good instructional design and a *methodology*.

1.3. FORMALISING, DESCRIBING AND STRUCTURING KNOWLEDGE AND SUPPORTING THE AUTHOR

As from the description above can be deduced, ISLEs will become complex environments that will contain large amounts of knowledge. The inventory of ISLE related information (de Jong, 1990a) provided a description of this information in *natural language*. There is, however, a need for a more structured and *more formal description* for at least two reasons. First, different components in an ISLE have to cooperate, which implies that it must be possible to *couple* the knowledge from different components. Second, a more structured description of knowledge may help to *reduce the gap* between natural language and programming language.

A second important conclusion is that most software tools merely help authors to implement what they have designed before they use the authoring tools. We feel that there is a need for helping authors in a conceptual way as well. Tait (1990) used the terms *execution tools* and *creation tools* to distinguish these two types of support tools. This implies that next to a structured, more formal language, there is a need for identifying 'conceptual units' in the ISLE related information.

Both aspects of structuring information are captured in what we have termed the 'formalisation'. So formalisation in our sense embraces both the use of a structured language for the description of the information and the identification of conceptual units in that information.

First we will briefly address the language aspect and then turn to the more conceptual level of formalisation: the identification of elements, relations and structures in the information.

2. Formalising ISLE related information; the syntax level

Knowledge, and also the ISLE-related knowledge, can be expressed in a number of different languages. The following types of languages can be distinguished:

- Natural language;
- (Semi-formal) structured languages;
- Logic and mathematics (genuine formal languages)
- Programming languages.

The knowledge that we gathered in our inventory reports is almost exclusively expressed in natural language. What we need is a more structured (semi-formal) language that enables us to represent the knowledge in a more precise and parsimonious way. This language should:

- (a) possess a limited number of basic concepts that provide significant expressive power (the principle of parsimony);
- (b) provide means for establishing clear communication among the components of an ISLE.

An ISLE is a heterogeneous set of cooperating components. It is unrealistic to expect a single representational scheme to satisfy all the needs. Moreover, despite the fact that the representational language is in itself without content, there is a relationship between what is to be expressed (the information) and the way it is expressed. Fernström, de Hoog, Hove, Humphreys, and Stokke (1987) have evaluated representation languages in relation to information and although their purpose was different from ours it is worth noting that they ended up devising their own representation method. The design of such a method (with complete syntax and consistent grammar) is a major task and is, we believe, outside the scope of SIMULATE. Consequently, for the design of our language we used methods of representation that already exists, and our analysis of the various representational methods commonly used (see, for example, Frost, 1986; Hull and King, 1988) has led us to decide to use only three schemes: frames, production rules, and functions.

The primitives of the language can be divided into elements (which may represent concepts, learning goals, variables and so on), relations (which may represent functions, procedures and the like) and structures, basic chunks of elements and relations.

The effort of creating a representation language is called '*formalisation*' in the project. The term formalisation should not be seen in a very strict sense regarding only logic and mathematics as formal languages. The aim of formalisation as described in this paper is to move away from the usable, but unsatisfactory formalisation (using a grain size that is too small) which expresses functionality as a computer program, and of the readily understandable but unprecise functionality as described in natural language and towards a formalisation which captures the same functionality in terms that more easily correspond with ideas and concepts that authors can be expected to bring to the task of creating ISLEs. We propose a 'formal language' that is still structured enough to be translatable, in principle, into a programming language. In this way a 'formal language' helps by reducing the gap between natural language and programming language and we see formalisation as a description of information at a level

that is situated somewhere in between natural language and programming language (see also Koper, 1989). A second important aspect of our formalisation aspect is to provide generic *elements*, *relations* and *structures*, thus furnishing the semantics of the language. In this way we will try to satisfy a third requirement of our formalisation effort:

- (c) facilitate unambiguous description.

We envisage these generic elements, relations and structures to support the authoring process to a great extent.

3. Formalisation and the authoring process, the semantic level

3.1. AUTHORING FOR CAI AND ICAI

In creating courseware authors take decisions in a number of areas: the domain, the curriculum, the interaction etc. In authoring for 'traditional' CAI these decisions are taken in 'one shot'; for example, authors create a curriculum in domain terms. So, when the author decides to change one aspect of the courseware, e.g. the curriculum structure, this means re-arrangement and rewriting of other aspects as well.

In ICAI (or ITS) a more modular approach is chosen. A now generally accepted set of modules is the domain, learner model, instruction and interface component. The idea is that if an author wants to change the domain, s/he can change it without having to change the other components. As for example Elsom-Cook and O'Malley (1990) point out, this ideal situation is hardly ever met.

Moreover, no high level tools for creating ICAI exist, so changes have to be generated by expert programmers. On the other hand, numerous tools exist to assist the author in creating CAI (see for example Barker, 1987). However, these tools only help the author to implement the courseware after all design decisions have been taken, which limits the support to the technical level.

The SIMULATE project embodies an attempt to create an authoring tool for intelligent systems that support the author not only at a *execution* level, but also at a *creation* level. Inspiration was found in a development methodology for general knowledge based systems.

3.2. MODEL BASED KBS DEVELOPMENT

The ISLEs as we envisage them are Intelligent Tutoring Systems and as such we may find relevant information in development methodologies for knowledge based systems. An influential methodology for the development of KBSs is KADS (see e.g. Breuker & Wielinga, 1985; Schreiber et al. 1989). The basic idea in KADS is to use prototypical models of information (knowledge) as an intermediate between knowledge as provided by an expert (and extracted in the knowledge acquisition process) and the coding of that knowledge in a knowledge based system. A central role is given to the *interpretation model*. An interpretation model consists of three layers. The first layer is the *inference layer*. At the inference layer an *epistemological description* of the to be modelled expert task is given. Basic entities at the inference layer are *metaclasses* and *knowledge sources*. A meta-class identifies the *role* a concept can play in the reasoning process (such as "hypothesis" and "observable"), the knowledge sources identify

Figure 1. An example of an inference structure (monitoring).

actions (such as "select" and "match") that can be executed on a metaclass, thus yielding new metaclasses. The idea is to provide the KBS builder with a library of prototypical combinations of metaclasses and knowledge sources from which s/he might choose and that s/he may combine to build a model task. A library of inference structures is given in Breuker et al. (1987). An example of such an inference structure for a prototypical task ("monitoring") is given in Figure 1. This example is taken from De Jong, De Hoog and Schreiber (1988) and was developed in the context of knowledge acquisition for a software project management system. Metaclasses are depicted as rectangles, knowledge sources as ovals.

Inference structures merely depict a 'logical combination' of knowledge sources and metaclasses. What is needed is a *task structure* (the second layer of the interpretation model) that defines which sequence of metaclasses and knowledge sources can be followed, and if different task structures exist, a *strategy* (the third layer) that describes when to choose which

task structure.

An important aspect of inference structures, like the one from Figure 1, is that *no actual domain terms are present*. Inference structures can be filled with appropriate domain concepts, and the same inference structure can thus be used for a number of different domains.

It is important to note that this means an even further separation of information than is normally done in ITS systems (see the preceding section).

3.3. MODELS FOR DESCRIBING ISLE RELATED INFORMATION

We have adopted the basic ideas from KADS on *model based design* and on *separation between factual domain information and epistemological descriptions* and made an adaptation to our specific aim, design of (or authoring for) intelligent tutoring systems and more specifically Intelligent Simulation Learning Environments².

The KADS approach, as briefly reviewed in the preceding section, was developed in the context of Knowledge Based Systems Design, and as such is mainly concerned with the *domain information* and the inference structure, which we will term *operational model*. In our context, however, we need to extend in two directions.

First, instruction may not only be concerned with teaching some kind of task (or operation) but also with teaching the more conceptual aspects (containing both static and dynamic characteristics) of a domain. This bipartition is generally recognised (see for example de Jong & Ferguson-Hessler, 1986) and also applied in the context of ITS development (Park & Seidel, 1990). It is therefore that we introduced a second 'epistemological level', the *conceptual model*³.

Second, in knowledge based systems, the domain information together with the operational model suffice to give a description of the task the KBS has to perform. In our context (which is instruction), however, this operational model (and the conceptual model) become the *object of instruction*.⁴ We therefore need additional models that use the conceptual and interpretation model for instruction. These additional models are the *instructional model*, the *learner model*, and the *interface model*.

Our current view of the different levels is elaborated in Table 1. The first level in Table 1 gives the domain information. This level contains a set of domain data.

The second level depicts the conceptual and the operational model. As all models in this framework, the conceptual and operational model are made up of the three basic entities from our language: elements, relations and structures. The table gives some examples of elements, relations and structures for the conceptual and operational models. For the operational model

²The framework depicted here presents our current view. It should however be stressed that our work on formalisation is still under progress.

³Our terminology may sometimes be confusing with regard to the terminology as used in KADS. However, we cannot avoid using old terms in a new way.

⁴In recent literature on KADS a similar notion can be found in what is called *modality* (de Greef, Breuker & de Jong, 1988; Schreiber, Wielinga, Hesketh & Lewis, 1990).

these have been borrowed from the KADS methodology (see e.g. Breuker et al., 1987). Conceptual and operational models also have to cooperate in a very delicate way, since the operational model may use information from the conceptual model.

In Table 1, we can easily recognise the different types of learning goals as distinguished in the SIMULATE project (see van Berkum & de Jong, 1990). As basic types of learning goals

TABLE 1. Modelling framework for ITS in general and ISLE in particular.

Domain information Set of domain data			
Conceptual model	Elements: <ul style="list-style-type: none"> • variable • parameter • concept • 	Relations: <ul style="list-style-type: none"> • is_a • is_part_of • threshold • asymptotic • 	Structures: <ul style="list-style-type: none"> • model {type A, type B, ..} • process {type A, type B, ..} •
Operational model	Elements: <ul style="list-style-type: none"> • hypothesis • finding • observable • follows • 	Relations: <ul style="list-style-type: none"> • select • classify • 	Structures: <ul style="list-style-type: none"> • fault diagnosis • monitoring •
Learner model	Elements: <ul style="list-style-type: none"> • Conceptual model • Operational model • 	Relations: <ul style="list-style-type: none"> • misconceptions • malrules • goals 	Structures: <ul style="list-style-type: none"> • misconceptual structures • goal structures •
Instructional model	Elements: <ul style="list-style-type: none"> • Conceptual model • Operational model • 	Relations: <ul style="list-style-type: none"> • prerequisite • simplification • example • 	Structures: <ul style="list-style-type: none"> • progressive implementation • organiser •
Interface model	Elements: <ul style="list-style-type: none"> • Conceptual model • Operational model • 	Relations: <ul style="list-style-type: none"> • temporal • spatial • 	Structures: <ul style="list-style-type: none"> • all kinds of typical interaction patterns •

we identified *conceptual knowledge* and *operational knowledge*. These two types of goals could be *more generic* or *more domain specific* or could be *more or less compiled*. When we have a generic goal emphasis is put on the conceptual or operational model as such, when the learning goal concerns domain specific knowledge these models are *instantiated* with domain information. Declarative or compiled knowledge calls for different structures in the two models.

At the third level in Table 1 we find the instructional, learner and interface model. The elements of these models are principally the conceptual and/or operational model. This can be either structures, elements or relations (or combinations) from these models. Characteristic for these models is that they have their own specific type of relations. These relations may have a

Figure 2. A multi-level view on ITSs in general and ISLEs particularly.

more 'conceptual nature' or a more 'operational' nature, the latter is similar to recognising specific 'tasks' within an ITS (see for examples Winkels, 1989). We are thinking of separating each of the three models in the third level into a 'conceptual' and an 'operational' part.

Possibly, the instructional, learner and interface models need their own types of elements, and, moreover, we may assume that, with a concrete description of a domain in terms of an actual conceptual and/or operational model, not all relations at the third level can be satisfied. Actual conceptual and operational models are defined in a parsimonious and 'context-free' way (they reflect the 'domain'), which may lead to including additional descriptions at the highest level (such as 'implicit' relations in simulation models (see van Joolingen & de Jong, 1990)). Of course these 'extra' features can be instantiated as well, which may call upon extra domain information. This multi-level structure is depicted in Figure 2.

Finally, we like to make two extra comments to this multi-level model approach. The framework as presented here gives a complete view on elements of ISLEs. As we stated before and as is detailed in Hijne and van Berkum (1990) we envisage an ISLE to exist of an ISLE shell and an ISLE knowledge base. Part of the functionality of an ISLE can be seen as invariable and as such will be placed in the ISLE shell. For example *control structures* that decide upon the interaction of different elements will be situated in the ISLE shell. Other

functionality will be part of the ISLE knowledge base.

Second, the models presented here all have the *domain* as their basis. The instructional, interface and learner model may also, however, have elements that are not specifically domain related. This is most clearly illustrated by learner attributes from the learner model in for example an attribute such as 'level of being systematic' (Self, Pengelly, & Twidale, 1990). These elements also need a structured ('formal') description.

3.4. THE PROCESS OF AUTHORING

The structures as identified in the formalisation work don't give a direct opportunity for building a software system such as an ISLE. However, the structures will be an important input for specifying building blocks an author may use for actually creating ISLEs. These basic building blocks (BBBs) form the software counterparts of the structures from Table 1. Although, one should strive to a close mapping of structures and building blocks, practice proves that this is *not straightforward and easy to do* (Schreiber et al., 1989).

Starting from a repertoire of domain independent building blocks, the authoring process can be divided into three stages: *selection*, *specialisation* and *instantiation*. After *selection* and *specialisation* the author fits together the chosen building blocks into a generic ISLE knowledge base, that is an ISLE knowledge base which is appropriate for a number of related contexts (where the context is determined by the domain, the learning goals, the learner population and the environment in which the ISLE is to be used). *Instantiation* takes the generic ISLE knowledge base and adds the domain terms to it, in this way turning it into a specific ISLE knowledge base. This is illustrated in Figure 3.

Figure 3. Authoring through selection, specialisation and instantiation.

In order to specialise a building block the author needs access to the representation language used to specify building blocks. This has some important consequences. The first of these is that the language cannot be esoteric and difficult to learn: authors are not software engineers. The second consequence is that support must be available for an author who wishes to engage in specialisation: this kind of support is not easy to provide and may be of the same order of complexity as provided in computer assisted software engineering (CASE). Thirdly, specialisation is only likely to be attractive to a subset of authors, in particular those with accumulated experience and expertise.

Instantiation is a different matter. The very nature of instantiation only requires the author to be knowledgeable about the syntax which determines the class of legal entities that may, for example, be used to fill a slot. This requires less support and even the least experienced of authors (provided they know the subject matter and understand the pedagogical requirements) will be able to make use of instantiation to produce a specific ISLE, relatively quickly.

4. An example of formalisation

In this section we present some concrete examples from our formalisation work. For this we will draw most heavily on the conceptual model because our progress in the most advanced at this point. We will also give an idea of a range of possible relations from an instructional model. All examples regard our main interest: the development of learning environments around *simulations*. At this moment we have concentrated at elements and relations and will turn to structures later on.

4.1. THE CONCEPTUAL MODEL

The model that drives computer simulations—in Van Joolingen and De Jong (1990) the term *runnable model* is introduced—is mostly given in terms of *variables*, describing a model *state*, and a set of rules, which determine the change of variables and their interrelationships. These rules can be influenced by parameters. For the conceptual model similar elements are defined. Elements in a conceptual model are of two main types: *variables* and *parameters*. Variables represent inputs and outputs of the simulation, as well as the state of this system. Some variables are closely linked to variables which are part of the runnable model, others represent more higher order views on the same model. Mostly, however, variables in the runnable model, which represent similar concepts will be described by only one variable in the conceptual model.

A variable in the conceptual model is described by the following attributes:

- a *name*
- a *value range*, which is the set of all possible values
- the dependency of *time*, which largely determines the characteristics of the variable
- a *default* initial value for the variable.

The attribute of value allows for both quantitative values or qualitative values with a very

distinct meaning, as well as also more descriptive qualitative values. Moreover, both types of values may be used for one variable at the same time. This means that a variable may have more than one range. This need for more than one range can be achieved by a simple inheritance mechanism which allows a variable frame to add a range of values to the range of the parent variable. This can only be provided if a mapping exists between the two ranges.

We have chosen a frame approach and the *variable frame* that we use for describing a variable is:

```
variable <name> [inherit <var-frame name>]
range <range>
time-dependency <y/n>
initial value <value>
.
```

Parameters are much like variables. They represent external influences on the simulated system. Therefore, the only difference between a parameter and a variable frame is the (initial) value slot. Since parameters are not changed by internal causes their value is constant, or, for dynamic parameters, has a fixed time dependency. This leads to the following parameter frame:

```
parameter <name> [inherit <par or var -frame name>]
range <range>
time-dependency <y/n>
value <value>
.
```

In order to describe relations between basic elements in the conceptual model the term *generic relation* has been introduced (Van Joolingen & De Jong 1990). With this concept it is possible to yield a *relation typology* that describes the characteristics of the relations between variables and parameters.

A generic relation is described in terms of:

- A *domain* and a *range* which both are value ranges of the variables involved.
- Its *type*, is the relation a *function* or not.
- *Parameters*, which can act within the relation.
- A *condition* which describes for which variable tuples the relation is true.

This leads to the following generic relation frame:

```
generic relation <name>
domain <any set>
range <any set>
function <y/n>
parameter <list of parameters>
condition <condition>
.
```

Now we will give two examples of generic relations, expressed in this frame language. These generic relations form one of the semantical aspects of our formalisation endeavour.

The first example is a generic relation that can hold between two variables (or one parameter and one variable) that have a range which is an ordered set of values. The relation states that if the input variable increases, the output variable also increases⁵.

```
generic relation M+
domain <ordered set>
range <ordered set>
function y
condition {R ∈ P(domain × range) | ∀ (a1, b1), (a2, b2) ∈ R,
      a1 > a2 : b1 > b2}
```

The second example expresses a characteristic that is often found in models. In the case of this example a variable is dependent of another one, the so called *threshold variable*. The interdependency of these variables is different for values of the threshold variable below or above a certain threshold value. In this generic threshold relation the behaviour below and above the threshold value is described by two different relations, which are 'parameters' to the threshold relation:

```
generic relation Threshold1
parameter t, R1, R2
condition { R1, R2 ∈ P(domain × range) | ∀ a ∈ domain: a < t :
      R1(A, B); a ≥ t : R2(A, B) }
```

Another case is when the output variable is the threshold variable itself. In that case the condition should replace a with b. This threshold relation will be called *Threshold*₀.⁶

The basic structure of the conceptual model is the *model*. This model will often be a submodel in a larger structure; the construction of these larger structures will not be discussed here (but see van Joolingen & de Jong, 1990). A model is a collection of variables and parameters and their interrelationships. The model frame looks like this:

⁵In these examples $\mathcal{P}(\text{domain} \times \text{range})$ denotes the set of all relations possible on the specified domain and range.

⁶We have defined a number of generic relations for the conceptual model (see van Joolingen & de Jong, 1990).

```

model <name> [ inherit <model name> ]
variables
  (list of the following structures:)
  <name> : <type> : <vartype>
parameters
  (list of the following:)
  <name> : <type>
relations
  (list of the following:)
  <type>[(parameters)](<varname>,<varname>) :
    [ explicit | implicit ]
.

```

The frames presented up until now had a generic character and were not filled with domain information. We will now give an example of *instantiation* of a conceptual model. The example that has been chosen is very simple: the relation between the added heat and temperature of a substance near its melting point. The relation between these variables is a double threshold relation. Below the melting point the temperature rises in a monotonic way with the added heat. As soon as the temperature reaches the melting point (the first threshold value) the temperature remains constant, until all material has become liquid, i.e. as soon as the energy added since the melting point had been reached, has crossed the second threshold, the melting energy. From that point on the temperature rises again, monotonically.

The example may not seem very revolutionary, but it can be used to show some interesting aspects of the domain representation mechanism. After having defined the parameters and variables the model reads as:

```

model melting
variables
  q : energy : input
  t : temperature : output
parameters
  melting_point : temperature
  melting_energy : energy
relations
  Threshold0 [melting_point, M+, Constant] (q,t)
  Threshold1 [melting_energy+q(t=melting_point), Constant,
    M+] (q,t)
.

```

This model is itself an specialisation of more generic models which include the above combination of two threshold relations. The model can be instantiated, e.g. by assigning specific values to the parameters, making the model specific for one particular substance. Another specialisation could be limiting the range of the input variable energy, e.g. allowing only positive values, meaning that energy can only be added, not extracted. Finally, also the relations that take part in the threshold relations can be specialised further, for example by replacing them with linear relations.

4.2. THE INSTRUCTIONAL MODEL

In this section we list a few examples of relations as we have currently identified for the instructional model. These relations are at the moment still defined in natural language. They are listed here to give an impression of possible instructional relations.

- *is_a_typical_situation_of*
For simulations it is sometimes important to know which situations are frequently found as situations in a real system (the real system can be some artifact or physical system). Variable or parameter values in the model are set to "typical values".
- *is_a_critical_situation_of*
A critical situation is a structure in which (selected) variables and parameters are set to values that are at the border of changing the characteristics of the model (so threshold variables are set to their critical values).
- *is_a_specific_type_of*
Models may differ in generality. In a more general model variables, parameters and relations each (or apart) are more global than in a more specific model (and as a consequence less precise). A *monotonic* relation is more global than a *monotonic increasing* relation.
- *is_a_simplification_of*
Simpler models leave out specific relations, variables or parameters from more complex models.
- *is_analogous_to*
Two structures are analogous when the structural properties of both models are the same. Structural properties refer to the types of variables and parameters and the types of (generic) relations between them as described in the cognitive model.

5. A model-based approach in a DELTA perspective

The structures of the identified models whose description will be the output of 'formalisation' are similar to the so-called 'half-fabricates' of Open System for Instructional Design (OS-ID) (see Derks & Bulthuis, 1990) which proposes the idea of an open workbench into which a variety of tools can be plugged during the development of instructional materials. In order to provide this integrating function OS-ID will need a very structured (formal) description of the knowledge base. Thus formalisation is a step towards OS-ID. Half-fabricates are seen as reusable, and the idea of predefined, reusable information chunks is not new in DELTA or elsewhere. Tait and France (1986) state "Formalisation of each learning activity will lead to the specification of a flexible 'environment'... [these] environments... will provide all the attributes of macros, formats, templates shells and so forth." Chiocciariello et al. (1989), for example, stress the idea of reusability of 'units', 'components' or 'learning resources'. In the DELTA project LEAST an attempt is made in standardising aspects of Open Learning Material (OLM) (see e.g. Clarke, 1990); this project also identifies 'primitives'.

Defining and formalising entities of Intelligent Simulation Learning Environments will be a major effort of which a first, and modest, attempt was outlined in this paper. Translating these

structured and formal descriptions into their software counterparts (basic building blocks) and incorporating these into functional designs of intelligent systems is even a big step further. These are all very valuable enterprises that are only partly feasible to perform within DELTA, and, therefore, certainly ask for something Beyond.

References

- Barker, P. (1987). *Authoring languages for CAL*. Houndsmill: Macmillan.
- Berkum, J.J.A. van, & Hijne, H., (1990). *The ISLE Functional Architecture*. Courseware Europe: DELTA Project SAFE Report SAFE/SIM/ WPIV/B-pap/islefa.
- van Berkum, J.A., & de Jong, T. (1990). Instructional environments for simulations. In: T. de Jong (Ed.), *Computer simulations in an instructional context*. DELTA project SAFE (P7061), deliverable SIM/21 (pp. 133-201). Eindhoven University of Technology: SAFE/SIM/WPI/EUT-rep/com.final.
- Breuker, J.A. & Wielinga, B.J. (1985). KADS: Structured knowledge acquisition for expert systems. *Proceedings of the fifth international workshop on expert systems and their applications*. (Avignon) France.
- Breuker, J.A., Wielinga, B., van Someren, M., de Hoog, R., Schreiber, G., de Greef, P., Wielermaker, J., Billaut, J.P., Davoodi, M., & Hayward, S. (1988). *Model driven knowledge acquisition: Interpretation models*. Deliverable A1. Esprit Project P1098. Amsterdam: UvA, Department of Social Science Informatics.
- Chiocciariello, A., Midoro, V., Olimpo, G., Persico, D., Sarti, L., Tavella, M., & Trentin, G. (1989). *A conceptual model for a database of learning material (DBLM)*. ITD/CNR: DELTA Project D1012, Deliverable 2.
- Clarke, G. (Ed) (1990). *User interface standards in learning environments*. MARI: DELTA Project LEAST, Draft deliverable 12.
- Derks, M., & Bulthuis, W. (1990). *A framework for an open tool environment*. Paper presented at the DELTA & Beyond conference. The Hague (the Netherlands), October 18-19.
- Elsom-Cook, M.T., & O'Malley, C.E. (1990). ECAL: Bridging the gap between CAL and intelligent tutoring systems. *Computers & Education*, 15, 69-81.
- Fernström, C., Hoog de, R., Hove, J. O., Humphreys, P., & Stokke, G. (1987). *A representation language for PIMS Part A: Evaluation*. PIMS Working Paper UVAPAPRRP001. University of Amsterdam.
- Frost, R.A. (1986). *Introduction to Knowledge Base Systems*. London: Collins.
- Greef, P. de, Breuker, J., & de Jong, T. (1988). *Modality. An analysis of functions, user control and communication in knowledge based systems*. Deliverable D6. Esprit Project P1098. Amsterdam: UvA, Department of Social Science Informatics.
- Hollan, J.D., Hutchins, E. L., & Weitzman, L. (1984). STEAMER: An interactive inspectable simulation-based training system. *AI Magazine*, 5(2), 15-27.
- Hull, R., & King, R. (1988). *Semantic database modelling: Survey, application and research issues*. ACM Computer Surveys, 201-261.
- Hijne, H., & van Berkum, J.A. (1990). *A functional architecture for intelligent simulation learning environments*. Paper presented at the DELTA & Beyond conference. October 18-19, The Hague (Netherlands).
- de Jong, T., (Ed) (1990a). *Computer Simulations in an Instructional Context*. Eindhoven University of Technology: DELTA Project SAFE Report SIM/21.
- de Jong, T., (1990b). *Providing support in learning from computer simulations; interface*

- aspects. Paper to present at the NATO ARW "Cognitive Modelling & Interactive Environments, Eindhoven (The Netherlands), October 1990.
- de Jong, T., & Ferguson-Hessler, M.G.M. (1986). Cognitive structures of good and poor novice problem solvers in physics. *Journal of Educational Psychology*, 78, 279-288.
- de Jong, T., de Hoog, R., & Schreiber, G. (1988). Knowledge acquisition for an integrated project management system. *Information Processing and Management*, 24(6), 681-691.
- de Jong, T., & Njoo, M. (1990). *Learning and Instruction with computer simulations: learning processes involved*. Paper presented at the NATO ARW "Computer-based Learning and Problem Solving", Leuven (Belgium) September 1990.
- van Joolingen, W.R., Castells, J., Abreu, J-L., Fabrega, A., & Olivera, M. (1990). Simulation construction tools. In K. Tait (Ed.), *Towards the specification of integrated tools for authors constructing simulation-based intelligent learning environments*. University of Leeds: DELTA Project SAFE Report SIM/22.
- van Joolingen, W.R. & de Jong, T. (1990). Formalisation of the domain representation for ISLEs. In T. de Jong & K. Tait (Eds.), *Towards formalising the components of an Intelligent Simulation Learning Environment*. DELTA project SAFE, deliverable SIM/11-18.
- Koper, E.J.R. (1989). *Inscript: een scriptaal voor het systematisch ontwerp van interactieve leersystemen* (Inscript: a scriptlanguage for the systematic design of interactive learning systems). OTIC Research Report No. 4. Heerlen (The Netherlands): Open University.
- Park, O., & Seidel, R.J. (1990). A multidisciplinary model for development of intelligent computer assisted instruction. *ETR&D*, 37, 72-80.
- Psotka, J., Massey, L.D., and Mutter, S.A. (Eds) (1988). *Intelligent Tutoring Systems: Lessons Learned*. Hillsdale NJ: Lawrence Erlbaum Associates.
- Schreiber, G. (Ed.). (1990). *A KADS approach to KBS design*. Deliverable B6, ESPRIT project P1098. Amsterdam: UvA Department of Social Science Informatics.
- Schreiber, G., Bredeweg, B., Davoodi, M., & Wielinga, B. (1987). *Towards a design methodology of KBS*. Deliverable D8, ESPRIT project P1098. Amsterdam: UvA Department of Social Science Informatics.
- Schreiber, G., Wielinga, B., Hesketh, P., Lewis, A. (1990). *A KADS design description language*. Deliverable B7, ESPRIT project P1098. Amsterdam: UvA Department of Social Science Informatics.
- Self, J., Pengelly, M., & Twidale, M. (1990). Formalising learner modelling tools for simulations. In T. de Jong & K. Tait (Eds.), *Towards formalising the components of an Intelligent Simulation Learning Environment*. DELTA project SAFE, deliverable SIM/11-18.
- Shute, V.J., & Glaser, R. (1990). A large-scale evaluation of an intelligent discovery world: Smithtown. *Interactive Learning Environments*, 1, 51-77.
- Tait, K. (Ed), (1990). *Towards the specification of integrated tools for authors constructing simulation-based intelligent learning environments*. University of Leeds: DELTA Project SAFE Report SIM/22.
- Tait K, & France, S.A. (1986). *DELTA: The Authoring Facility* (published as Chapter V of Cazala et al.(Eds), 1987, DELTA Initial Studies, Les Editions du Logiciel d'Enseignement, Paris
- White, B.Y., & Frederiksen, J.R. (1987). *Causal Model Progressions as a Foundation for Intelligent Learning Environments* (Report No. 6686). Cambridge, MA: BBN Laboratories.
- Winkels, R. (1989). *A new framework for describing and designing intelligent tutoring systems*. In: Proceedings of the CEPES/UNESCO symposium on "The advent of AI in Higher Education" Prague, 23-25 October 1989. Springer Verlag.