SMISLE: System for Multimedia Integrated Simulation Learning Environments

Ton de Jong¹, Wouter van Joolingen¹, David Scott², Robert de Hoog³,
Laurent Lapied⁴, & Robert Valent⁵

¹ Faculty of Educational Science and Technology, University of Twente, The Netherlands; ² Marconi Simulation, United Kingdom; ³ Department of Social Science Informatics, University of Amsterdam, The Netherlands; ⁴ Framentec-Cognitech, France; ⁵ Engineering Systems International, France.

Abstract

The SMISLE project (System for Multimedia Integrated Simulation Learning Environments) has two main objectives. First, it aims to define exploratory learning environments based on simulations that incorporate instructional support for learners in such a way that effective and efficient learning will result. Second, it aims to provide authors of these simulation learning environments with an authoring toolkit that not only presents technical, but also conceptual support. Providing *support to the learner* can be done in many different ways. The project started with an inventory of potential instructional support measures and selected four types of measures that now have been implemented: progressive model implementation, assignments, explanations, and hypothesis scratchpads. The simulation environments that incorporate these support measures are designed around five different models each carrying a specific function. The *runnable model* is an efficient representation of the domain that will make the simulation run; the *cognitive model* is the representation of the domain that is tailored to learning and instruction; the *instructional model* incorporates the instructional support; the *learner model* keeps track of knowledge and characteristics of the learner; and the *interface model* decides upon the appearance of the simulation environment to the student. Together these models form the resulting application for the learner, which is called a MISLE (Multimedia Integrated Simulation Learning Environment). The main task of an author is to create the different models in the MISLE, with the exception of the runnable model which is automatically generated from the cognitive model. Creating the different models essentially means that an author has to select, specialise and instantiate generic *building blocks* (that can be regarded as generic templates) that are offered in *libraries of building blocks*. For each of the models there is a separate library of building blocks and a set of *dedicated editors* for specialising and instantiating the building blocks. Additionally, authors are guided through the authoring process by a *methodology* and they have access to *instructional advice* which provides them with ideas on which instructional support measures to apply.

1 INTRODUCTION

In contemporary theories of learning and instruction, learners are no longer seen as persons who copy knowledge that is presented to them by a teacher, but as active agents in the learning process. This view has led to new approaches to the design of instructional environments. One of these new approaches is *scientific discovery learning* or *exploratory learning*, in which the learner is not given direct instruction about the domain to be mastered, but is invited to discover the important concepts and relations in the domain him or herself.

The main assumption behind the idea of exploratory learning is that knowledge discovered by the learner will be based more firmly in the learner's knowledge base and have stronger links with the learner's prior knowledge than knowledge that is just told. Knowledge that is discovered is seen as more intuitive, qualitative knowledge. Also the skills needed for the exploration itself are seen as important skills, since they may help a person in dealing with unfamiliar situations in a more effective way.

These notions have received particular attention through the introduction of *computer simulations* as an instructional device. Computer simulations can play an important role as a training device for reasons of safety, economy and for social reasons (de Jong, 1991). More important, however, is that simulations are well suited to exploratory or discovery learning since they contain a model that has to be discovered by the learner.

Computer simulations are quite popular in instruction (de Jong, van Andel, Leiblum, & Mirande, 1992), but in spite of their popularity there is no conclusive evidence of their effectiveness and efficiency, with some studies showing effects (Shute & Glaser, 1990; Grimes & Wiley, 1990; Faryniarz & Lockwood, 1992), whereas others fail to find advantages of simulations (Carlsen & Andre, 1992; Rivers & Vockell, 1987; de Jong, de Hoog, & de Vries, 1993). The reason for the possible ineffectiveness of computer simulations is that learners, while interacting with a simulation, encounter difficulties which they cannot overcome on their own. These problems concern, for example, the adequate execution of exploratory processes such as formulating hypotheses (van Joolingen & de Jong, 1991; Njoo & de Jong, 1993a,b), the switching between hypotheses and experiments (Klahr & Dunbar, 1988) and the burden of regulatory processes such as planning, keeping track of what has been done and checking (Shute & Glaser, 1990). A possible solution for these problems is to provide learners with support in addition to a pure simulation.

A second problem with the use of computer simulations in instruction concerns another actor: the author. The above mentioned inventory (de Jong, van Andel, Leiblum, & Mirande, 1992) showed that simulations are almost always created using general programming languages. Languages for non-programmers, such as authoring languages, are generally employed for creating drills and tutorials. This indicates a potential market for *authoring tools for simulations*. Taking into account

our assertion that simulations should be surrounded with instructional support for the learner, these authoring tools for simulations should also assist the author in creating the additional learner support. Since simulations and instructional support for learners will be new areas for many potential authors, adequate authoring tools should not only provide technical support, as all contemporary authoring languages do, but also need to *provide the author with conceptual design support*.

The two issues discussed above – providing learners with simulations that have integrated instructional support and providing (non-programmer) authors with authoring tools that include both technical and conceptual design support¹ – are the key elements of the SMISLE project. The following sections outline the approach and achievements of the SMISLE (System for Multimedia Integrated Simulation Learning Environments) project. Section 2 gives an overall impression of the types of simulation learning environments that can be created using the authoring environment that is under construction; the SMISLE toolkit². Section 3 presents the general ‘architecture’ of the applications for learners (the so-called MISLEs, which stands for Multimedia Integrated Simulation Learning Environments). The authoring process with the SMISLE toolkit consists of selecting, specialising and instantiating building blocks from libraries of building blocks. Section 4 describes the building blocks that are available in the SMISLE libraries and Section 5 subsequently presents the functionality that is present in the SMISLE toolkit for browsing, inspecting, selecting, specialising, and instantiating generic building blocks. Section 5 also describes two features of the SMISLE toolkit that assist authors in their work: an authoring methodology and pedagogical advice.

2 (MULTIMEDIA) INTEGRATED SIMULATION LEARNING ENVIRONMENTS

A (Multimedia) Integrated Simulation Learning Environment (from now on we will use the short term *MISLE*) presents the learner with a simulation and instructional support. Currently four types of instructional support have been selected. These are *model progression* which means that a learner may work through a succession of more complex and more precise domain models, *assignments* that help the learner to set targets while working with the simulation, *explanations* offering the learner direct access to domain information, and *hypothesis scratchpads* that help the learner in formulating hypotheses. Figure 1 displays the interface of a typical MISLE. It shows a number of windows of which the upper left window contains the simulation

¹ The author we have in mind for working with the SMISLE authoring toolkit is a domain expert not necessarily having programming experience.

² The functionality and interfaces as presented in the present paper are based on SMISLE version 1.11 which was released in February 1994. The SMISLE project will continue until the end of March 1995 and will use results of ongoing evaluation studies to adapt functionalities and interfaces. SMISLE 1.11 is built in Smalltalk 80 and VisualWorks™ (ParcPlace Systems) and is portable over Windows, Macintosh and Unix platforms.

itself (with input variables that can be changed by the learner and a display of output variables); the bottom left window gives the learner access to overall control functions; the remaining windows are each associated with one of the four types of instructional support. In the interface we have included an example of a simple simulation on controlling the heating of a house; input variables are the leaking to the outside world, the heating of the house itself, opening or closing a window, and the outside temperature, the output variable is the inside temperature. Learners working with a MISLE will normally switch regularly between operating the simulation window and use of instructional support, either on their own or on system initiative. Each of the instructional support measures is treated in detail in Section 4.2.

Figure 1 An example of a MISLE interface

3 THE ARCHITECTURE OF MISLES

This section describes the architecture that underlies simulation learning environments (MISLEs) that can be created with the SMISLE authoring environment. We have divided a MISLE into a number of functionally different units that are called MISLE models. We distinguish the following models:

- *Runnable model*
In a MISLE the runnable model will make the simulation run. The runnable model is itself an abstract description of a particular domain. The runnable model has as its objective to enable an optimal fast and efficient simulation. It will, therefore, often have a numerical character, but it may also be a qualitative causal model.
- *Instructional model*
The instructional model contains the instructional function of the MISLE. It carries all the instructional measures available to the learner, such as (for example) the possibility of presenting explanations.
- *Learner model*
The learner model is able to store (or infer) information regarding a learner. This information can then be used for activating instructional measures.
- *Interface model*
The interface model contains all kinds of graphical objects, windows etc. enabling the learner to interact with the MISLE.

This four-partition follows a division into components which is commonly used in describing Intelligent Tutoring Systems (see for example Wenger, 1987). The domain related model of these four components (the runnable model), however, will, quite often, not be a good basis for instruction. First, we will not, for example, want learners to master complex sets of differential equations. Also, since the runnable model needs to be efficient, (abstract) variables, that are necessary for the learner to master, might be absent in the runnable model (see van Joolingen & de Jong, 1992). Second, as the 'domain representation' will be used by the other models (instructional, learner, and interface model) the runnable model probably would not contain sufficient information for the other models to function. For example, if the instructional model prescribes the generation of a causal explanation, the cognitive model will have to be structured in such a way to allow this kind of explanation.

To solve the first problem, we introduce the concepts of *conceptual model* and *operational model* (de Jong, Tait, & van Joolingen, 1992). These two models are the instructional domain representations, one for conceptual domains (domains in which no inherent procedure is present, for example electrical circuits), the other, required in addition, for operational domains (domains that have an inherent procedure, for example flying a plane). These two models represent what we want the learner to know or master at the end of the training session. Conceptual models may contain different models of the same system; for instance one model may give detailed information on the behaviour of a system whilst another may describe its causal structure.

To solve the second problem mentioned we introduce *extensions* of the conceptual model and operational model that provide the (extra) necessary information for the instructional, learner and interface models to operate. These extensions are added to the conceptual and operational models on the basis of requirements from the instructional, learner and interface model.

Together we have called the conceptual model, operational model and their extensions the *cognitive model*. This cognitive model plays a central role in SMISLE applications. Figure 2 displays a diagram of the different models.

Figure 2 *Different models as identified in the SMISLE project*

4 THE SMISLE LIBRARIES

For a specific MISLE, authors will create the different MISLE models on the basis of building blocks from *libraries of building blocks* offered in the SMISLE authoring toolkit. For each of the MISLE models identified in the previous section, a library of building blocks will be available to the author. Only the runnable model is not created directly by the author but is generated from the cognitive model. Figure 3 gives a simple illustration of the relation between the libraries (part of the SMISLE authoring toolkit) and the resulting application (the MISLE).

Clearly a project like SMISLE will not be able to deliver an authoring toolkit that can be used for creating MISLEs of any kind, including all types of domains and all types of instructional measure etc. The present section outlines the choices (and thus restrictions) that have been made in the project for the different models in the MISLE, which, naturally, comes down to the choice of building blocks to be made available in the libraries of the MISLE.

Figure 3 Schematic relation between model libraries and models

4.1 The cognitive model library

The cognitive model which is present in an application (MISLE) fulfils a dual role. First, it provides a description of the domain which is tailored to fit its use in instruction. This has two aspects: the representation of the knowledge that the learner should acquire in the course of exploring the simulation, and the provision of support to the other components of the MISLE (in particular the instructional model) so that they can function properly. For instance, the latter could include methods for determining the complexity of a domain model which is important in instructional theory. Second, the runnable model employed by the MISLE is generated automatically from this more qualitative description of the domain, rather than being directly encoded in a special purpose simulation language or a general programming language.

In concept at least, a *cognitive model* is composed of a *conceptual model*, an *operational model*, and some *extensions*. The extensions are present purely to support the other components of the MISLE. The conceptual and operational models are the counterparts of the conceptual and operational domains which were described in Section 3. Three libraries are provided to support the construction of the cognitive model. The *functional block library* and the *petri net library* are used to build the conceptual model, whilst the *operational model library* is used to build the operational model.

4.1.1 *The conceptual model library*

We have decided that the types of systemic model that the formalism must be capable of representing are *lumped parameter* and the *process interaction view of discrete event simulation*, possibly coupled together. There are separate libraries for these two sorts of model namely the *functional block library* and the *petrinet library* respectively.

The functional block library

The formalism usually used to describe lumped parameter models is that of ordinary differential equations. Systems of differential equations may be represented by networks of *functional units* such as adders, multipliers, integrators, etc. So, one way in which models may be constructed is by connecting together such units. In general, these functional units are multi-directional - for example, an adder can compute the value on one of its terminals given the values on the other two - but this is not the case for all of them (such as explicit sources and sinks). A network representation has the advantages of reducing the scope for syntactic errors in the description, and eliminating the need both for the explicit introduction of temporary variables and for the equations to be ordered.

Any given type of functional block has a fixed number of *terminals*. A terminal may be an *input* or *output* terminal, or it may be *bi-directional*. Terminals are connected by links. A link may connect two bi-directional terminals (in which case the link is represented by a simple line), or an output terminal to an input terminal (in which case the link is represented by an arrow).

A functional block may be composed of other functional blocks, called *sub-blocks*, in which case it is a *compound* (or *complex*, or *non-primitive*) *functional block*. The terminals on a compound block correspond to the free terminals of its sub-blocks. A *free terminal* is a terminal which is not linked to any other terminal.

A subset of the library of functional blocks allows systems to be modelled using the formalism of *bond graphs* (Karnopp, Margolis, & Rosenberg, 1990). When it is appropriate to model a system in terms of energy flowing between its constituents this provides a concise graphical notation. In this representation systems are described as collections of elements which may store energy, transform energy, dissipate energy, etc. These elements are called *ports*. A 1-port has just one energy flow in or out of it, a 2-port has two energy flows, etc. The basic n-port elements are shown in Table 1. *Bonds*, denoted by half arrows, indicate the sign convention for the directions of the energy flows. A pair of variables is associated with each bond: an effort variable, e , and a flow variable, f . The product of these variables has the dimensions of power. Sometimes one wishes to neglect the energy flow associated with an interaction. This may be the case when observing or controlling a system. In this case the interaction is indicated by a full arrow which is called an *active bond*. The basic 2-port elements provided in the library are the modulated transformer and modulated gyrator which have active bonds.

Table 1 *The symbols, names, and relations of the basic n-port elements*

Symbols	Names	Relations
$\frac{e}{f} \rightarrow R$	linear resistor	$e = Rf$
$\frac{e}{f} \rightarrow C$	linear capacitor	$f = dq/dt, q = Ce$
$\frac{e}{f} \rightarrow I$	linear inertia	$e = dp/dt, p = If$
$Se \rightarrow$	effort source	$e(t)$ given, $f(t)$ arbitrary
$Sf \rightarrow$	flow source	$f(t)$ given, $e(t)$ arbitrary
$\begin{array}{ccc} & m & \\ & \downarrow & \\ \frac{e1}{f1} \rightarrow & MTF & \rightarrow \frac{e2}{f2} \end{array}$	modulated transformer	$e1 = m.e2$ $m.f1 = f2$
$\begin{array}{ccc} & r & \\ & \downarrow & \\ \frac{e1}{f1} \rightarrow & MGY & \rightarrow \frac{e2}{f2} \end{array}$	modulated gyrator	$e1 = r.f2$ $r.f1 = e2$
$\begin{array}{ccc} & e2 \downarrow f2 & \\ & \downarrow & \\ \frac{e1}{f1} \rightarrow & 0 & \leftarrow \frac{e3}{f3} \end{array}$	0-junction	$e1 = e2 = e3$ $f1 + f2 + f3 = 0$
$\begin{array}{ccc} & e2 \downarrow f2 & \\ & \downarrow & \\ \frac{e1}{f1} \rightarrow & 1 & \leftarrow \frac{e3}{f3} \end{array}$	1-junction	$f1 = f2 = f3$ $e1 + e2 + e3 = 0$

One n-port may be joined to another n-port by selecting one bond from each (they must not have been used already to create a connection) and identifying them. Corresponding relationships are established between the effort and flow variables of the two bonds. More complicated connections may be established via the 3-port junctions. Figure 4 shows representations of a simple harmonic oscillator and a damped harmonic oscillator using the bond graph notation.

The bond graph notation is not supported directly by the modelling tool. Instead bond graphs appear as complex building blocks in the library of functional blocks (i.e. they are constructed from simpler functional blocks) as is illustrated by Figure 5

Figure 4 *A simple oscillator (left) and a damped oscillator (right) in bond graph notation*

Figure 5 *The implementation of various n-ports in terms of functional blocks*

in which the networks corresponding to the resistor and transformer are shown. Figure 6 shows the functional block representations of the bond graphs of Figure 4. Note that the graphical notation of bond graphs is not fully supported. In particular the modelling tool will not automatically take care of the conventions for power flow. For this reason new 2-ports called ‘0-inverter’ and ‘1-inverter’ have been introduced so that the direction of a power flow may be reversed. A similar approach has been taken in DYMOLA (Cellier, 1991).

The petri net library

The formalism used to implement discrete event models is that of *extended petri nets*. This formalism is also used to represent operational expertise and is described in more detail in the next section. The petri net formalism may be used to describe the behaviour of adders, multipliers, etc., so a fully integrated description of

Figure 6 *A simple oscillator (left) and a damped oscillator (right) in functional block notation*

behaviour may be obtained which mixes lumped parameter and discrete event models. Functional units such as adders are represented in the library of extended petri nets as complex transitions. Figure 7 gives an example. The fact that basic functional blocks may be represented in this way is important for the working of the system, but is largely irrelevant as far as the author is concerned. Extended petri nets may be used to represent causal structures which include inhibition.

Figure 7 *The petri net for 'adder'*

4.1.2 *The operational model library*

The formalism used to model operational expertise is that of *extended petri nets*. Standard petri nets will not be described here other than to say that they are composed of places (represented by circles) and transitions (represented by bars) connected by directed arcs (represented by arrows), but a description may be found in (Petersen, 1981). However, the additional features that we use are described in this section. An operational model must have an associated conceptual model, for it describes how some system is manipulated. We shall use petri nets to represent tasks with their places being used to represent directly observable aspects of the state of the system and their transitions being used to represent actions performed by the operator on directly accessible parts of the system.

Standard petri nets may be used to model parallelism and synchronisation – which is obviously required in the representation of operational expertise. Also, they can express the logical constraints on the ordering of actions, and show the results of those actions. However, given that we wish to represent operational expertise, they cannot express certain ideas which are important for us, such as:

1. that an action cannot (or should not) occur if some condition holds;
2. that the operator may interact with a continuous system;
3. that the operator may have to interact with a dynamic system, which involves several issues:
 - the operator may have to wait for the system to reach a certain state;
 - one action may have to (or should) occur a certain time after another action, i.e. the operator must (should) wait;
 - an action whilst being treated as elementary in most respects nevertheless actually consumes some time.
4. that the operator may have to perform a computation;
5. that a task may be decomposed into sub-tasks.

Consequently some additional features are required, the principal ones being:

1. *Inhibitor arcs*

An *inhibitor* arc may run from a place to a transition. The presence of a token in the place prevents the transition from being enabled.

2. *Concrete actions*

Transitions have already been associated, conceptually, with actions of the operator. We now allow the association of a *concrete action* with a transition. Such an action either sets the value of some input variable of the system or gets the value of some output variable of the system.

3. *Coloured tokens*

Concrete actions associated with transitions may be parametrised, and they may return results which may be used as arguments to other actions. These values are carried around the net by *coloured tokens*.

4. *Conditional transitions*

A *delayed transition* has a time interval associated with it. If a delayed transition is selected for firing then it does not actually fire until the specified period of simulated time has elapsed.

5 *Transitions dependent on events in the associated conceptual model*

The enabling conditions for *state dependent* transitions depend on logical expressions which involve the values of one or two output variables of the associated conceptual model. Such a transition is not enabled unless the logical expression evaluates to true and it satisfies the usual enabling condition.

6. *Arithmetic transitions*

An *arithmetic transition* has an associated arithmetic operator and produces an output token whose value is determined by applying the operator to the values of its input tokens. The number of operands required by an operator is fixed and their order is usually significant. In order to take this into account new features, called *terminals*, are introduced on all transitions. A terminal is either an *input terminal* or an *output terminal*. Arcs no longer connect places to transitions (or vice versa), instead they connect places to terminals (or vice versa). A terminal may be connected to at most one place, but a place may be connected to any number of terminals. An input terminal may have an arc coming into it, but not one leaving it, whilst an output terminal may have an arc leaving it, but not one coming into it.

7. *Compound transitions*

A *compound transition* has a petri net associated with it which will be referred to as its *sub-net*. The terminals on a compound transition are determined by its sub-net. Each terminal corresponds to a *free terminal* on a transition of the sub-net. A free terminal is a terminal which is not linked to anything by an arc. When a non-primitive transition fires it does not immediately put out tokens at its output terminals. Instead execution of the sub-net is started.

4.2 The instructional model library

Providing learners with instructional support in exploratory environments can be done in many ways. The project started with an inventory which resulted in a list of 23 different types of instructional measures (de Jong, van Joolingen, Pieters, van der Hulst, & de Hoog, 1992), and this list certainly is not exhaustive. From this list of instructional support measures four types of support were selected to be included in the instructional model library for SMISLE. To select these measures we used a number of criteria which included that measures should be applicable to conceptual and operational domains and also that they should form a more or less coherent combination. This coherence was achieved by adopting two basic approaches to instruction. One is the *mental model approach* as it is reflected in the work by White and Frederiksen (1989; 1990). The most essential characteristic of this approach is that a domain is learned by exposure to a sequence of models that gradually progress to some target model. The second approach is the *cognitive apprenticeship approach*

as it is described by Collins, Brown, and Newman (1989). The mental model approach was developed in the context of teaching conceptual domains, the cognitive apprenticeship approach is applicable to operational domains. Both approaches have in common that students' faults are prevented. The latter characteristic of both approaches means that there will be no heavy burden on a 'learner model'. The specific instructional support measures that resulted are: model progression, assignments, explanations, and hypothesis scratchpads. Each of these support measures is discussed in detail in one of the following subsections.

4.2.1 *Model progression*

The basic idea of offering model progression in a simulation environment is to gradually unfold the properties of the domain to the learner, by offering a sequence of models. Model progression is used to avoid a confrontation of learners with a model that is too complex with respect to their prior knowledge. Model progression has been applied in systems such as QUEST (White & Fredriksen, 1989; 1990), NEWTON (Teodoro, 1992), and DIBI (Plötzner & Spada, 1992). In the SMISLE instructional model library we provide possibilities for:

- *Simple to complex ordering.*
Simple to complex progression means that models with only a few variables develop towards models with a large number of variables. For example, a simulation environment about mechanics can start with frictionless motion in one dimension and move to motion with friction in three dimensions.
- *Changing the 'order' of the model.*
We make a distinction between zero and higher order models, differing along the dimension of precision, i.e. going from qualitative descriptions to quantitative ones.
- *Changing perspective on the model.*
This includes offering different ways to describe the same domain, for example using different variables. An important difference between this dimension and the ones listed above is that there is no ordering implied by the different perspectives; this means that the author has to supply this ordering.
- *From partial task to complete task.*
Learners start with performing parts of the complete task and gradually they get control of the complete task. This type of model progression refers to operational domains, whereas the first three refer to conceptual domains.

4.2.2 *Assignments*

Confronted with an educational simulation the learner's general (often not stated) goal is to 'discover' the model behind the simulation. Such an 'assignment' may be too vague for a learner to work with, leading to undirected exploratory behaviour.

Therefore, we may introduce assignments in order to give learners some grip and to direct them through the model. Assignments can be given before the simulation starts, before a set of simulation runs, or at any moment in the simulation. We have distinguished six types of assignment:

- *Investigation*
Investigation assignments prompt the learner to find the relation between specific variables.
- *Optimisation*
In an optimisation assignment the learner is asked to change values for input variables in such a way that the output variable(s) acquire certain values that can be described as maximal, minimal, or optimal.
- *Fault diagnosis*
In fault diagnosis the learner is confronted with a model that does not behave as it should; the assignment is to find the faulty relation.
- *Specification (prediction)*
In specification assignments the learner is asked to predict (calculate) the unknown value for some output variable.
- *Explication*
In explication assignments the learner is asked to give an explication for a specific phenomenon in terms of causal relations. This means that the MISLE shows the learner input values together with output values from the simulation and the learner has to give an account of what is displayed.
- *(Normal) operation*
An important class of operational tasks concerns the normal operation of devices. This involves the starting, maintaining and shut down of processes. In these types of assignments output variables are specified, and the learner has to provide input for some of the input variables, whereas other input variables are under control of the system. For example: keeping a plane in the air has specified output variables (distance from the ground > 0); the learner has to provide values for input variables (the position of the wing flaps for example); and some input variables (wind direction) are under control of the system.

Normal operation is an assignment associated with operational domains, optimisation assignments can refer to both conceptual and operational domains, and the remaining assignments refer to conceptual domains³. Three of these assignments (investigation, explication, and fault diagnosis) focus *directly* at relations in the model, the other three do so *implicitly* by asking for specific values of input or output variables. Table 2 gives a structured overview of assignments present in the SMISLE instructional model library.

³ Fault diagnosis assignments only refer to operational domains when the operational task *is* fault diagnosis, in that case, however, fault diagnosis is normal operation.

Table 2 *Structured overview of the different types of assignments*

Specification	Normal operation	Optimisation
Varl ₁ : specified Varl ₂ : specified Varl ₃ : specified Varl ₄ : specified VarO ₁ : ?? VarO ₂ : ??	Varl ₁ : to be varied Varl ₂ : to be varied Varl ₃ : varied Varl ₄ : varied VarO ₁ : specified VarO ₂ : specified	Varl ₁ : ?? Varl ₂ : ?? Varl ₃ : ?? Varl ₄ : ?? VarO ₁ : specified VarO ₂ : specified
Investigation	Explicitation	Fault diagnosis
Varl ₁ : to be varied Varl ₂ : to be varied Varl ₃ : to be varied Varl ₄ : to be varied VarO ₁ : displayed VarO ₂ : displayed R(Varl ₁ ,VarO ₁): ??	Varl ₁ : varied Varl ₂ : varied Varl ₃ : varied Varl ₄ : varied VarO ₁ : displayed VarO ₂ : displayed R(Varl ₁ ,VarO ₁): ??	Varl ₁ : to be varied Varl ₂ : to be varied Varl ₃ : to be varied Varl ₄ : to be varied VarO ₁ : displayed VarO ₂ : displayed R(Varl ₁ ,VarO ₁): specified R(Varl ₂ ,VarO ₁): specified R _f (Varl _x , VarO _y): ??
'specified' and 'varied' are author actions 'displayed' is system action 'to be varied' is learner action '??' indicates 'values' (which can be single values or ranges of values) to be given by learner		

4.2.3 Explanations

Explanations are instructional measures by which learners are offered information on concepts that play a role in the simulation model, or they are offered relations between variables directly. Explanations are used to present learners with (lacking) prior knowledge that they need for a fruitful exploration (e.g. definitions of variables) or to give learners relations between variables directly when they fail to discover them. Explanation as an instructional support measure with simulations has been used for example by Shute (1991) and Leutner (1993). In a MISLE the following types of explanation can be offered:

- *Class dependent*
A 'class-dependent' view gives the place of a concept in a class hierarchy.

- *Structural*
A structural explanation lists the super-part or sub-parts of a concept.
- *Functional*
In functional explanations the role of an object in a process is presented.
- *Modulatory*
A modulatory explanation lists how ‘one object or process affects (or is affected by) another process or object’. In SMISLE 1.11 only one modulatory relation which is the ‘causes’ relation is provided.
- *Attributional*
An attributional explanation lists the attributes of a concept.
- *Covariance*
Two concepts may show covariance without having a modulatory relation. Different types of covariance relations exist such as ‘monotonic increasing’, ‘feedback’, and ‘asymptotical’. A covariance explanation informs the learner on the relevant covariance relation between two variables.
- *Analogy*
An analogy presents the learner with a similar variable or relation as the one under study, but now in a different (type of) system, or it refers to some real world application
- *Additional information*
In additional information the author may put any information he or she likes.
- *Condition-action pairs*
This explanation tells the learner which conditions have to be fulfilled for a certain (learner suggested) action to be performed, or which action is preferable under certain conditions. This explanation refers to operational domains only.

The author may select specific types of explanation that he or she wants to use in the MISLE that is created. There is, of course a restriction that the domain should allow for certain explanations (for example, if no causal relations exist, no modulatory explanations can be given). For each explanation the learner indicates the object he or she wants an explanation on and selects the types of explanation wanted from a list of available explanations.

4.2.4 *Hypothesis scratchpads*

Research has shown that creating hypotheses is one of the most difficult aspects of discovery learning (Klahr & Dunbar, 1988; Njoo & de Jong, 1993a; 1993b; Chinn & Brewer, 1993; Dunbar, 1993). Hypothesis scratchpads are learner instruments designed to support the learner in stating hypotheses about the simulation. The idea of hypothesis scratchpads is based on the ‘hypothesis menu’ in systems such as Smithtown (Shute & Glaser, 1990), Voltaville (Glaser, Raghavan, & Schauble, 1988) and Refract (Reimann, 1991). The role of hypothesis scratchpads has been empirically investigated by van Joolingen and de Jong (1991; 1993) and van Joolingen (1993). Hypothesis scratchpads contain the elements necessary for

composing hypotheses. For conceptual domains these elements are: variables, relations and conditions. Hypothesis scratchpads for operational domains comprise similar elements. Instead of variables, however, these operational scratchpads contain ‘actions’, which themselves consist of variables and value changes (e.g. increase Var₁). Conditions in the conceptual scratchpad refer to constraints on the validity of the hypothesis, in the operational scratchpad they may be enabling or triggering conditions.

Learners can use the hypothesis scratchpad to compose a hypothesis from the elements that are provided: save the hypothesis which will put it on the hypothesis list; load a hypothesis from the list and allow further editing; mark the truth value of a hypothesis [true/false/unknown]; and, finally, mark a hypothesis as being tested [yes/no]. Figure 8 gives an example of a hypothesis scratchpad in this case for the conceptual domain of the heater that was displayed in Figure 1.

Figure 8 Example of a hypothesis scratchpad.

4.2.5 *An example of an instructional measure building block*

The preceding sections gave an overview of all generic building blocks in the SMISLE libraries that are at the author's disposal for creating instructional support measures. In this section we provide a more detailed example of one of the instructional measures present as a building block in the SMISLE libraries: a *normal operation assignment*. The purpose of this type of assignment is that the learner practice a task in the simulation environment. For example, in the case of a flight simulator, a normal operation assignment would mean operating the simulation of the aircraft under specified circumstances. Typically, a number of normal operation assignments together will allow practice in a number of varying conditions.

In a MISLE a normal operation assignment (when selected by the learner or activated by the MISLE) sets up the simulation in the state from which the assignment should start. The learner is told the goal of the assignment (e.g. take off, fly to position X and land safely) and starts operating the simulation. During the activity of the normal operation assignment, the MISLE checks if the task is carried out correctly. This is done by checking *constraints* which were set by the author. These constraints specify the boundaries within which the learner may operate, e.g., in the case of the plane: the height is greater than zero. Once a constraint is broken, (height < 0, the plane crashes) the learner's attempt is terminated and an appropriate message is displayed. Alternatively, the learner can reach the goal (safely on the ground in place X), in which case the assignment is terminated successfully.

The functionality for setting up the simulation, checking the constraints and informing learners of their success or failure are present in the library building block representing a normal operation assignment. In order to include a normal operation assignment in a MISLE, all the author has to do is create a copy of the building block and specify the initial state for the assignment, the constraints and the target of the assignment. E.g. for the assignment for flying the aeroplane to X, the author specifies the position of the plane at the start of the assignment, the fact that the height should be always greater than zero and the coordinates of the target position.

Apart from the details of a normal operation assignment, the author also has to specify when the assignment may become active. This is done by attaching *enabling conditions* to the assignments, which specify the state of the MISLE at which the assignment can become active. MISLEs that are created with SMISLE 1.11 put most initiative (for selecting assignments, asking explanations etc.) in the hands of the learner. The only system initiative is the progression to a subsequent model level and the activation of assignments and explanations. Although these actions are initiated by the learner, the system may define minimum requirements that should be met (in terms of achieved assignments) before a specific instructional measure may be activated.

In Section 5.3 the tool (the instructional model editor) will be discussed which the author uses in the process of specifying instructional measures and the tool for specifying enabling conditions.

4.3 The learner model library

Since, as was explained in Section 4.2.5, in the current state of SMISLE most initiative is on the learner's side, SMISLE 1.11 only needs limited learner modelling capabilities. In the (near) future we foresee more sophisticated system initiative which partly will be based on a learner model. For creating this learner model a MISLE keeps a record of relevant events. These events fall in two main categories: interaction with the simulation and interaction with the instructional support. The history is recorded as a sequence of these events, but the data are also collected in more global records, like records of the number of variables varied, the total number of interactions etc. SMISLE enables the author to determine which of these events should be included in the learner model. Events that are recorded can be used for creating enabling conditions (see Section 5.3).

4.4 The interface model library

The learner interface communicates both the results of the simulation and the information regarding the instructional support to the learner. For this purpose there are six windows on a typical MISLE screen: one simulation window, four windows, each associated with one type of instructional support (model progression, assignments, explanations and hypothesis scratchpads) and a generic control window (see Figure 1). The four instructional measure windows display the current active instructional measure of the associated type and, if applicable, control the interaction of the learner with instructional support. So, the hypothesis scratchpad window allows the learner to state hypotheses, whereas the assignment window allows the learner to select assignment as well as provide answers to them. These windows are predefined and are not changed by the author and hence have no associated library in SMISLE. The same holds for the control window.

The interior of the simulation window is defined by the author, who can access a library of interface building blocks. The building blocks in this library are visible elements of a display which can be coupled to variables in the cognitive model. The SMISLE interface library includes graphs, tables, gauges, dials, thermometers, buttons, numerical displays, etc. Also, the author can import (static) graphics from other applications to be displayed in the interface window and create sequences of bitmaps to generate animations. In Section 5.5, the tools used to create the simulation interface and to access the library of interface building blocks are described.

5 BUILDING A MISLE

In Section 4 we have presented an overview of the generic building blocks an author has available. The present section describes the SMISLE toolkit that an author may use for browsing, inspecting, selecting and instantiating the building blocks, in this way creating the different models that together constitute a MISLE. First, we will describe how an author may browse, inspect and select building blocks from the libraries (Section 5.1) and then we will detail the dedicated editors for instantiating building blocks in the Sections 5.2 to 5.5. Finally, authors are supported in the SMISLE toolkit by an authoring methodology (Section 5.6) and pedagogical advice (Section 5.7)

5.1 Browsing, inspecting, and selecting building blocks

Building blocks that have been selected by the author can be specialised (by changing the value of specific slots) or instantiated (which means that domain variables are included in the building blocks). Authors have access to the libraries of building blocks for selecting a building block in three different ways. First, they can choose the option 'Create Cognitive ('Instructional', 'Learner', or 'Interface') Model', in which case they are presented with lists of relevant building blocks from which they may choose. Browsing building blocks is done by scrolling through the list. Second, authors may choose the option 'Libraries' in which case they are presented with a tree representation of all libraries with basic building blocks as the leaves of the tree. After selecting a basic building block (e.g. investigation assignment) the associated editor is shown so that the author may inspect the contents of the building block. Third, authors may perform a certain task by choosing the option 'Authoring', in which case they are shown a task decomposition in the form of a tree, of which basic tasks (e.g. create investigation assignment) form the leaf ends.

5.2 Creating the cognitive model

The task of creating the cognitive model consists of taking building blocks from one of the three cognitive model libraries (functional block, petri-net, or operational) and linking these together. A network editor will soon be available for this (and the models will be displayed in formats similar to those of Figure 6 and 7), but in SMISLE 1.11 the cognitive model editor still has textual interfaces, one for each of the model types. Figure 9 displays the interface of the functional block editor. It provides a number of lists, displaying the building blocks present in the library, those already present in the conceptual model, the free terminals, the connected terminals and the links. The operational and petri-net editor for SMISLE 1.11 look similar but also have a list displaying the places present in the model.

Figure 9 The Functional block editor

A building block is added by selecting it in the library, and pressing the ‘Add Block’ button. Then the terminals of this building block can be inspected, renamed, changed, and linked with a few mouseclicks. In the case of the petri-net based editors links can only be created between a terminal and a place or vice versa, in the functional block editor, terminals can be linked directly.

A functional block model can be compiled into a petri-net model. The compilation process is necessary for creating a runnable model, but it also allows the model to be edited further at the petri-net level. This can be useful to add functionality which the functional block formalism does not provide, such as discrete event simulation (see Section 4.1.1).

5.3 Creating the instructional model

Figure 10 shows the instructional model editor. This editor provides a number of lists which contain the instructional building blocks which already have been selected and instantiated by the author. The author can add an instructional building block by selecting it from the library. Then a *building block editor* can be opened on the building block selected.

Figure 10 The instructional model editor

Figure 11 shows an example of a building block editor for a normal operation assignment which was discussed in Section 4.2.5. The building block editor allows authors to fill in the details of the instructional measure for their particular situation, in the case of the normal operation assignment: an initial state, constraints to hold during the performance of the assignment and a target state to signal successful behaviour.

Assignment editor

Name

Optimize 1

Interface Model

HeatingUnit

Maximum Number of Attempts

10

Description

Keep the temperature between 19 and 21 centigrades for 10 minutes

Initial State

label	value
H	0
integrator.z	20
window	0
k	1

Optimisation Assignment

Target State

label	minimum	maximum
time	10	
integrator.z		
Tin		

Constraints

label	minimum	maximum
time		20
Tin	19	21
integrator.z		

OK

Advice

Cancel

Figure 11 Example of a dedicated building block editor

For each type of instructional measure a dedicated building block editor is available. The general lay-out of all these editors is similar but, of course, they differ in the nature of the details to be filled in by the author.

Another task for the author is to specify the conditions under which the learner may select specific instructional measures. For example it may be specified that assignment B can only be selected after assignment A has been completed. A condition editor is available for this task. On this editor, the author can specify the state of instructional measures, which should hold at the moment the instructional measure may be activated, as a logical tree. Figure 12 shows a sequence of actions to define the enabling conditions for an instructional measure.

Figure 12 The editor for creating enabling conditions

5.4 Creating the learner model

The learner model itself is not created by the author him or herself, but evolves as a function of the behaviour of the learner. The only task for the author is to specify the event types that have to be recorded in the history of the MISLE-learner interaction. The author selects these event types from a complete list that is accessible from the SMISLE main menu.

5.5 Creating the interface model

For creating the interface model, or, more specifically, the interior of the simulation window, the interface editor of the Visualworks™ environment was reused, adapted, and extended. Figure 13 shows the editing screen that the author sees and uses. On the left the library of interface building blocks is displayed, on the right the simulation window that is being created. The author selects interface building blocks from the library and is able to position and size them with the mouse in the simulation window.

After an interface element has been added to the simulation window a *properties editor* can be opened on it in order to fill in the specific details of the building block (see Figure 14). The properties editor is the equivalent of the building block editor for the instructional model. The most important element to be edited in the properties editor is the link to the cognitive model: an interface element represents a variable in

Figure 13 The interface editor

Figure 14 An example of a properties editor

the simulation. On the properties editor a list of variables in the cognitive model is automatically available from which the author may select. The properties editor is also used to specify properties like the minimum and maximum value for a slider, the properties of the axes of a graph, etc.

When the simulation window is completed it is installed in the MISLE, which means that the links with the cognitive model are instantiated and the simulation is ready to run. After installation it can immediately be tested, in which case the author sees and is able to interact with the simulation window as the learner will see it.

5.6 Methodology

From the previous sections it can be concluded that building a MISLE with the SMISLE environment is a complex task. This complexity addresses three levels:

- *Task level*

At this level the author has to decide which general tasks will be performed. A general task is a larger unit of work that has some meaning for the author. It is assumed that the author plans the work of designing a MISLE at this level. A comprehensive task tree is identified that covers a wide range of author concerns. Examples of tasks are: determine low-level learning goals, configure instructional support, determine need for learning environment. Only a subset of these tasks can be carried out with the SMISLE environment, but it is important to represent the notion that building a MISLE involves more issues than those present in the workbench.

- *Configuration level*

For those tasks which support is provided by the workbench, the author has to deal with a standard sequence of actions that is the same for all types of supported tasks. This sequence consists of three steps:

- ♦ select a building block from the relevant library;
- ♦ instantiate the building block for the domain and learner under consideration;
- ♦ integrate the building block into the existing set of building blocks.

As an example take the set of assignment types shown in Table 2. The author has to select the required assignment from the set (e.g. optimisation), give 'names' to the abstract variables (e.g. VarI₁ becomes 'amount of money') and 'values' to the output variables (e.g. VarO₁ becomes 'product mix realised') and finally integrate them with the other elements that are already part of the MISLE.

- *Instrument level*

The sequence identified at the configuration level must be carried out with the author instruments/tools available in the SMISLE environment. Thus the author has to master this tool, which requires a well designed author interface.

The goal of the methodology is to support and guide the author in mastering this complex task. This is mainly done by considering the entire authoring process as a

kind of constrained design activity. The constraints involved are of different types that are linked to the levels described above:

- *Strategic constraints* are derived from ‘common practice’. Authors have already been developing simulation systems for a long time and by tapping their expertise heuristics can be generated to steer the less experienced author. An example of such a heuristic is: if the target application is in an operational domain and the required fidelity level is high, work out the interface first before trying to instantiate instructional measures. The rationale for this heuristic is that what can be done with the interface may determine the content of the instructional model to a great extent. It is obvious that constraints of this type are offered to the author as suggestions that can be ignored. In the SMISLE project research is being carried out in order to discover the most salient heuristics.
- *Consistency constraints* that follow from the necessity to keep the MISLE that is being developed consistent. The author can effect changes in one component of a model that will lead to inconsistencies with the state of other models. As an example we can take the author who has decided on three levels of model progression but tries later in the authoring process to define assignments for four different levels. The SMISLE toolkit must check the work of the author for these unintended mistakes. It will give a ‘warning’ to the author, but will leave the correction to the author because it is conceivable that the author wants to have this inconsistency temporarily.
- *Logical constraints* arising from the dependencies between the models and the building blocks. Examples of these constraints are:
 - ♦ assignments can only be instantiated when a part of the domain model has been defined;
 - ♦ before implementing model progression the progression relations have to be identified in the cognitive model.

The main focus of the authoring methodology is on the constraints. Apart from these constraints the author also has to take into account pedagogical principles for designing educationally sound simulations. These will be most of the time related to specific properties of the domain and the target learners. These constraints are covered by author advice (see Section 5.7).

We have analysed the authoring task and produced initial task decomposition, based on interviews with experienced authors. This will result in a tailored process model for authoring that will act as a high level structure for guiding the work of the author. This process model can be seen as one of the ‘top level drivers’ of the SMISLE toolkit (see Section 5.1) as well as the ‘watchdog’ that focuses the constraints mentioned above. The methodology will become apparent to the author partly as a paper document, partly through constraints implemented in the SMISLE toolkit (through ‘greying out’ unavailable author actions) and partly as messages on the screen.

5.7 Pedagogical advice

The typical author using the SMISLE authoring toolkit is a domain expert who does not necessarily have programming and educational experience. To cope with the latter, the SMISLE toolkit includes an *advice* module. This module provides the author with hints and background information, necessary for making decisions on the choice and instantiation of instructional measures.

The SMISLE advice module consists of four main parts. The first part contains information on each of the individual instructional measures present in the SMISLE libraries. Each measure is explained in detail and specific situations in which the measures can be of use, as well as information necessary for instantiation are given.

The second part of the advice module takes the characteristics of the domain and target learner population as a starting point. The author provides the advice module with information concerning the specific situation (i.e. characteristics of the learners and the domain), on the basis of which the advice module generates suggestions for the selection of instructional measures. Figure 15 shows how the author can enter learner and domain characteristics.

Figure 15 The advice instructional measures dialogue screen

The third part contains general background knowledge on discovery learning and simulations as a learning environment. Currently this module has about 150 screens on such topics as: characteristics of simulations, problems that learners encounter in discovery learning, theories of discovery learning, successful simulation projects etc. The rationale for including this module is that authors may want to learn more about the ideas behind SMISLE, both out of interest and for justifying their design decisions.

Finally, the advice module includes a glossary, containing definitions of terms that occur in the SMISLE toolkit. This glossary is accessible both as a separate module and from the positions in the advice module where the glossary terms occur.

Figure 16 presents a screendump from the SMISLE advice module where the author reads advice on a normal operation assignment.

The author has access to advice through a special advice option in the SMISLE main menu, but may also jump quickly to a relevant part of advice from any other part of the authoring process by selecting an ‘advice’ button.

Figure 16 View on the SMISLE advice module

6 CURRENT STATUS AND FUTURE DEVELOPMENTS

In this paper we have described integrated simulation learning environments as they can be created with SMISLE 1.11, and the SMISLE toolkit 1.11 itself. As was pointed out, SMISLE 1.11 is an intermediate prototype of the SMISLE toolkit that was released at the end of the project in March 1995. At the moment two types of evaluation study with SMISLE 1.11 are carried out and the results will provide feedback for making improvements in the SMISLE toolkit.

The first type of evaluation studies concerns the authoring toolkit itself. Authors who differ in educational and domain expertise perform the complete authoring cycle with SMISLE 1.11 or are presented partial tasks (such as building the instructional support measures around an existing cognitive model). Data are gathered by logging

the author's actions and by an in depth observation (through thinking aloud techniques and interviews) of the authoring process. These data will be used to get indications of unclarities and inconsistencies in the current prototype, and to assess the author's needs in order to decide upon improvements of the toolkit's functionality.

The second type of evaluation is an evaluation of MISLEs. A total of five experiments that basically aim at assessing the effectiveness and efficiency of MISLEs as compared to simulations not embedded in instructional support. Also, an assessment of the effect of the separate SMISLE support measures (assignments, explanations, model progression, and hypothesis scratchpads) on the learning process and resulting knowledge is being made. The domains that are included in this evaluation are three conceptual domains from physics (oscillatory motion, collision and transmission lines) and two operational domains (the start-up procedure of a hydrogen purification unit of an ethylene plant and the modelling of contacting bodies). Experiments are conducted at the Universities of Twente (The Netherlands), Murcia (Spain), and Kiel (Germany), and at Engineering Systems International (France) and Exxon (United Kingdom). Evaluation studies all follow an experimental set-up in which different MISLE arrangements are compared.

Regardless of the outcomes of the evaluation studies some improvements of the SMISLE prototype are already under development. One of them is the development of a graphical interface to the cognitive modelling tools as already explained in Section 5.2. A second improvement is the introduction of a more flexible sequence of authoring actions as defined at the task level described in Section 5.6. At present, the sequence of authoring the different models is fixed, it starts with the cognitive model and ends with the instructional support measures. A pre-pilot evaluation with a limited number of authors pointed out that a number of authors prefer to start the development of a MISLE with the simulation interface instead of the cognitive model. A so-called 'agenda mechanism' will be developed that allows for the introduction of 'dummy' variables in the interface that later on will be replaced by variables from the cognitive model. A third improvement is that we will provide the author with 'author instruments' that will assist the author in performing specific authoring tasks. At the moment such an instrument for analysing the domain for creating different forms of model progression is ready as a prototype and will be tested soon.

REFERENCES

- Carlsen, D.D., & Andre, T. (1992). Use of a microcomputer simulation and conceptual change text to overcome students preconceptions about electric circuits. *Journal of Computer-Based Instruction*, 19, 105-109.
- Cellier, F.E. (1991). *Continuous System Modeling*. Berlin: Springer-Verlag, 274-282.

- Chinn, C.A., & Brewer, W.F. (1993). The role of anomalous data in knowledge acquisition: A theoretical framework and implications for science instruction. *Review of Educational Research*, 63, 1-51.
- Collins, A., Brown, J.S., & Newman, S.E. (1989). Cognitive apprenticeship: Teaching the crafts of reading, writing, and mathematics. In L.B. Resnick (Ed.), *Knowing, learning, and instruction: Essays in honor of Robert Glaser*. Hillsdale, NJ: Lawrence Erlbaum.
- Dunbar, K. (1993). Concept discovery in a scientific domain. *Cognitive Science*, 17, 397-434.
- Faryniarz, J.V., & Lockwood, L.G. (1992). Effectiveness of microcomputer simulations in stimulating environmental problem solving by community college students. *Journal of Research in Science Teaching*, 29, 453-470.
- Glaser, R., Raghavan, K., & Schauble, L. (1988). Voltaville, a discovery environment to explore the laws of DC circuits. *Proceedings of the ITS-88* (pp. 61-66). Montreal, Canada.
- Grimes, P.W., & Wiley, T.E. (1990). The effectiveness of microcomputer simulations in the principles of economics course. *Computers & Education*, 14, 81-86.
- Jong, T. de (1991). Learning and instruction with computer simulations. *Education & Computing*, 6, 217-229.
- Jong, T. de, Andel, J. van, Leiblum, M., & Mirande, M. (1992). Computer assisted learning in higher education in the Netherlands, a review of findings. *Computers & Education*, 19, 381-386.
- Jong, T. de, Hoog, R. de, & Vries, F. de (1992). Coping with complex environments: The effects of navigation support and a transparent interface on learning with a computer simulation. *International Journal of Man-Machine Studies*, 39, 621-639.
- Jong, T. de, Joolingen, W.R. van, Pieters, J.M., Hulst, A. van der, & Hoog, R. de (1992). *Instructional support for simulations: overview, criteria and selection*. DELTA project SMISLE, Deliverable D02. University of Twente, Department of Education.
- Jong, T. de, Tait, K., & Joolingen, W.R. van (1992). Authoring for intelligent simulation based instruction: a model based approach. In S.A. Cerri & J. Whiting (Eds.) *Learning Technology in the European Communities* (pp. 619-637). Dordrecht: Kluwer Academic Publishers.
- Joolingen, W.R. van (1993). *Understanding and facilitating discovery learning in computer-based simulation environments*. PhD Thesis. Eindhoven: Eindhoven University of Technology.
- Joolingen, W.R. van, & Jong, T. de (1991). Supporting hypothesis generation by learners exploring an interactive computer simulation. *Instructional Science*, 20, 389-404.
- Joolingen, W.R. van, & Jong, T. de (1992). Modelling domain knowledge for Intelligent Simulation Learning Environments. *Computers & Education*, 18, 29-38.

- Joolingen, W.R. van, & Jong, T. de (1993). Exploring a domain through a computer simulation: traversing variable and relation space with the help of a hypothesis scratchpad. In D. Towne, T. de Jong & H. Spada (Eds.) *Simulation-based experiential learning* (pp. 191-206). Berlin: Springer.
- Karnopp, D.C., Margolis, D. L., & Rosenberg, R. C. (1990). *System Dynamics: A Unified Approach (2nd. Ed.)*. John Wiley & Sons.
- Klahr, D., & Dunbar, K. (1988). Dual space search during scientific reasoning. *Cognitive Science*, 12, 1-48.
- Leutner, D. (1993). Guided discovery learning with computer-based simulation games: effects of adaptive and non-adaptive instructional support. *Learning and Instruction*, 3, 113-132.
- Njoo, M., & Jong, T. de (1993a). Exploratory learning with a computer simulation for control theory: Learning processes and instructional support. *Journal of Research in Science Teaching*, 30, 821-844.
- Njoo, M. & Jong, T. de (1993b). Supporting exploratory learning by offering structured overviews of hypotheses. In D. Towne, T. de Jong & H. Spada (Eds.) *Simulation-based experiential learning* (pp. 207-225). Berlin: Springer.
- Petersen, J. L. (1981). *Petri Net Theory and the Modeling of Systems*. Prentice-Hall.
- Plötzner, R., & Spada, H. (1992). Analysis-based learning on multiple levels of mental domain representation. In E. de Corte, M. Linn, H. Mandl, & L. Verschaffel (Eds.) *Computer-based learning environments and problem solving* (pp. 103-129). Berlin: Springer.
- Reimann, P. (1991). Detecting functional relations in a computerized discovery environment. *Learning and Instruction*, 1, 45-65.
- Rivers, R.H., & Vockell, E. (1987). Computer simulations to stimulate scientific problem solving. *Journal of Research in Science Teaching*, 24, 403-415.
- Scott, D., Hulst, A. van der, & Hoog, R. de (1992). *The Library Description Language*. DELTA deliverable, Project SMISLE (D2007). Edinburgh: Marconi Simulation.
- Shute, V.J. (1991). *A comparison of learning environments: All that glitters* Paper presented at the American Educational Research Association (AERA) Annual Meeting, Chicago, USA.
- Shute, V.J., & Glaser, R. (1990). A large-scale evaluation of an intelligent discovery world: Smithtown. *Interactive Learning Environments*, 1, 51-77.
- Teodoro, V.D. (1992). Direct manipulation of physical concepts in a computerized exploratory laboratory. In E. de Corte, M. Linn, H. Mandl & L. Verschaffel (Eds.), *Computer-based learning environments and problem solving* (NATO ASI series F: Computer and Systems Series) (pp. 445-465). Berlin: Springer.
- Wenger, E. (1987). *Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge*. Los Altos, CA: Morgan Kaufmann.
- White, B.Y., & Frederiksen, J.R. (1989). Causal models as intelligent learning environments for science and engineering education. *Applied Artificial Intelligence*, 3(2-3), 83-106.

White, B.Y., & Frederiksen, J.R. (1990). Causal model progressions as a foundation for intelligent learning environments. *Artificial Intelligence*, 42, 99-157.

Acknowledgement: The work presented here is carried out under the DELTA programme of the EC as project D2007. We gratefully acknowledge the contributions of the following, present and former, colleagues from the project: Simon King, Daniel Delmas, Catherine Loprieno, Christian Canella, Jean-Marc Loingtier (Framentec-Cognitech), Anja van de Hulst, Michiel Kuiper, Bert Bredeweg (UvA), Jules Pieters, Janine Swaak, Frank van Doorn (UoT), Ken Horne, Joe Brough (Marconi), Jean-Louis Gregis, Andre Alusse, Christophe Janvier (ESI), Hermann Härtel (IPN), Ernesto Martin (Murcia). SMISLE 1.11 has been implemented by Andre Alusse, Christophe Janvier, Jean-Marc Loingtier, Wouter van Joolingen, and David Scott.