Brno University of Technology
Faculty of Information Technology
Department of Intelligent Systems

An Environment for Modelling of Heterogeneous Systems

Prostředí pro modelování heterogenních systémů

CONDENSED VERSION OF DOCTORAL THESIS

Author: Ing. Martin Hrubý

Field of study: Information technology
Advisor: Doc. Ing. Zdena Rábová, CSc.
Opponent: Prof. Ing. Jiří Šafařík, CSc.
Opponent: Prof. Ing. Ivo Vondrák, CSc.
State doctoral exam: June 24, 2002
Submitted: March, 2004

Abstract

This work is oriented on methods and tools for building models of complex, large and heterogeneous systems. Their creation is associated with searching some suitable methods and tools allowing design and realization of models in various application environments. Modelling methods and tools reflect evolution of programming and simulation languages for many past years. For this reason, I have been inspired with philosophy and various clones of the object oriented programming which is now implemented in many interesting object oriented languages.

Analysis of heterogeneous systems was the starting point of this thesis – we split the modelled system to couple of sub-systems and we describe and solve each of them with its most natural and standard approach. We get so a set of sub-models for later connection into a whole heterogeneous model. Heterogeneous approach in modelling can significantly decrease complexity of models. Realization of sub-model interconnection and providing their cooperation is not a trivial problem. My approach of solving that is one of the original contributions of this thesis. The interconnection is based on a communication net called AIN (Automatic Information Net) which also defines semantics of a new simulation language called HELEF (**He**terogeneous **L**anguage **E**nvironment, **F**inal version). HELEF itself is the original contribution of the thesis too.

Key words: modelling and simulation, heterogeneous systems, heterogeneous models, interconnection, object orientation

Contents

Chapter 1

Introduction

This work is oriented on methods and tools for building models of complex, large and heterogeneous systems. Their creation is associated with searching some suitable methods and tools allowing design and realization of models in various application environments. Modelling methods and tools reflect evolution of programming and simulation languages for many past years. For this reason, I have been inspired with philosophy and various clones of the object oriented programming which is now implemented in many interesting object oriented languages.

Analysis of heterogeneous systems was the starting point of this thesis – we split the modelled system to couple of sub-systems and we describe and solve each of them with its most natural and standard approach. We get so a set of sub-models for later connection into a whole heterogeneous model. Realization of sub-model interconnection and providing their cooperation is not a trivial problem. My approach of solving that is one of the original contributions of this thesis. The interconnection is based on a communication net called AIN (Automatic Information Net) which also defines semantics of a new simulation language called HELEF (**He**terogeneous **L**anguage **E**nvironment, **F**inal version). HELEF itself is the original contribution of the thesis too.

The thesis has eight chapters and two appendix chapters.

The second chapter is an extended introduction. It contains a description of some selected object oriented languages. Languages Simula 67, Smalltalk, Self, Beta and PNtalk had a very big influence on the HELEF language created during my own work. The second chapter introduces their main advantages to the object oriented technology and emphasizes their special features and approaches – terms like persistence, interactivity and non-class object orientation (OO without classes).

The third chapter is an analyse of heterogeneous systems and models. It demonstrates increase of work productivity in modelling and programming when done in heterogeneous design. Solving the heterogeneity relates to identifying attributes common for all sub-systems which can be used as a communication interface. End of the chapter drafts a typical heterogeneous system architecture.

The fourth chapter is an essence of this thesis and it contains its main original contribution. It defines two aspects of any computation flow – data items and procedures (elementary actions). Those two aspects are called *data simulation object* and *process simulation object* here. Simulation object are linked to a special net called AIN by their data and logic relations. Created AIN net represents a standardized form of any sub-system in heterogeneous systems generally – it is a paradigm common for all sub-models included in the heterogeneous model and so all sub-models can be integrated to a working aggregate. Well known DEVS formalism dispose a similar feature too and the chapter ends with AIN and DEVS compare.

The method based on the AIN net solves methodological and functional interconnection of sub-systems (their technical wiring can be realized by CORBA platform, for example). *The fifth chapter* is a discussion about abstraction of reality into the model form. It demonstrates a meaning of term *models re-usability* based on structuring of models to so-called *environments* containing *properties* and *entities*. The chapter results with a certain method of object oriented formalizing of real-world.

Conclusions of the fourth and fifth chapters put a concept of new heterogeneous simulation language called

HELEF. HELEF is an object oriented simulation language with semantics based on the AIN net holding a hierarchy of environment, entity and property objects. The language itself is documented in *the sixth chapter* and is defined only briefly to keep this thesis in certain size. Description assumes basic knowledge of other object oriented languages.

Last part of the thesis (*the seventh chapter and appendix*) brings a couple of commented examples of typical modelling situations which are now coded in the HELEF language and one greater model as well.

Chapter 2

Research goals

There are three main goals to solve in my thesis. All are connected with the methodology of heterogeneous modelling and all lead to making a *general heterogeneous modelling environment*.

1. *Creating a concept and methodology of making heterogeneous models.* Heterogeneous models (multi-formalism models) are constructed of sub-models where every sub-model is based on generally different language paradigm, formalism or tool. Interconnecting the particular sub-models is a serious problem. Submitted thesis brings a solution of their interconnecting which is based on the AIN universal communication platform.

2. *Creating a new methodology of real-world objects modelling.* This thesis brings a certain analysis of structure and behavior of the reality. A higher level of abstraction in description of the real-world objects is the second contribution of this thesis. This also cohere with some re-usability of already made models.

3. *Design of a new simulation language resulting from the first and second paragraph.* Presented language (HELEF) is a new simulation language which may be a future advantage in the computer modelling.

Chapter 3

Current state of research in Heterogeneous and Object Oriented Modelling

Heterogeneous modelling [Fishwick TR, Hrubý 02, Hrubý MOS01] splits a given system to various characteristic classes – sub-systems, so that it is possible to model each class individually employing its characteristic formalism. Resulting sub-models may be encapsulated, instantiable and they can communicate each other. It is reasonable to combine the heterogeneous and object oriented styles of model design.

This work was inspired with an idea of choosing a suitable method for a given problem – we may say – by a resistance which the chosen formalism (a simulation language, for example) generate when we attempt to use it to code the problem. We probably chose that most suitable approach with the lowest level of resistance. Certain tasks have simply their characteristic methods of solving. We will probably use Petri net to describe processes and their mutual synchronization. We model logic tasks with some logic programming language, like Prolog is. We use a continuous simulation language to solve differential equations and so on. Those methods and tools are *single-purpose* and we have to *combine* them to make an *universal modelling system*.

3.1 Heterogeneity

Prof. Tuncer Ören[1] suggested in his paper [Ören 02] *Future of Modelling and Simulation: Some Development Areas* the future steps in development of modelling and simulation science. He put the heterogeneous methods on the first priority. In the category *Science, methodology and technology of modelling and simulation*, he enumerated these terms (only the relevant ones are mentioned):

1. Multi-models – group of very close modules of a model. The simulation run switches between particular modules so that one of them is always active.
2. Multiaspect, multistage, multiperspective, multiresolution, and multiparadigm modelling – multiaspect model is a special case of a multimodel where more modules can exist simultaneously. In *Multiresolution* modelling, we make a set of different models of a given problem – each model represents a different level of resolution (granularity, abstraction, precision). User selects one of them regarding his requirements.
3. Variable structure models.
4. Mixed formalism simulation – simulation is connected with an additional (different) type of knowledge processing (for example – an expert system giving some simulation parameters).
5. Specification languages and environments for inter-operability – inter-operability (cooperation, integration) is an important feature in integration of simulation models. The experience show that combing the existing program systems is more effective then building the new ones.

[1] Ören, T. I. - Professor Emeritus, Ottawa Center of the McLeod Institute of Simulation Sciences of the SCS, University of Ottawa, Ontario, Canada.

As we can see, there is a huge range of specific approaches using modularity, integrating a knowledge base, concurrent and interactive simulations, inter-operability and specification (interconnecting) languages of high abstraction. The scientific literature also mentions a multi-formalism term which is probably closest to our view of the heterogeneity. The article refers the HLA platform as a suitable tool for implementation of communicating modules.

The HLA and CORBA platforms

There are some interconnection platforms for programming distributed computings. Those system architectures (HLA, CORBA) [Buss 98] are usually also heterogeneous in meaning of interconnecting the various systems (at level of programming languages, for example).

These platforms physically transport some information from a system to another system. An A application is going to send a message to B application through a special interface: the interface converts the message to some universal format (including the binary level compatibility) and sends that to the opposite interface on the other side. The opposite interface adapts the message to a form acceptable for the B application. But the HLA and CORBA platforms (compared with a general heterogeneity):

- do not bring a general methodology of making a heterogeneous model.
- limit the set of pluggable sub-systems to a relatively small group of supported languages and development environments.

HLA and CORBA can serve as a stabilized and well developed communication layer for some higher heterogeneous technology. Unifying the binary data level, message formats and net protocols are necessary requirements of such a physical communication. The XML language is another possible way of sharing information among the sub-systems.

The computer system compatibility

Let us extend the application of heterogeneity to wide range of computer systems and especially to:

- *Operating systems*, where the compatibility among the different systems is a really important requirement.
- *Applications* in some operating system where the interconnection is still not satisfactory. The inter-application data transfer is very limited – in fact, it is only a text copy-paste facility. Data can be transmitted for example with the XML standard, but mostly if the both sides and prepared for this communication (similar function is supported by OLE and COM technology).

Implementing all systems in one language or paradigm would be the final solution. The Smalltalk and Self programming environments leads to this situations as they attempt to be an operating system. The world of unified programming language is illusory and probably even not acceptable. Making the applications with a built-in heterogeneous interface would be a good compromise.

Multi-paradigm languages

In the current time, so-called multi-paradigm languages are just being developed. These languages support more programming paradigms (logic, functional, constraint-bases, object oriented, procedural, parallel). Ole Madsen (the Beta language co-author) wrote in his article [Madsen 00]: "Our goal in the programming languages research and development should be in integrating of the best concepts and construction of various programming paradigms".

In most of cases, a multi-paradigm language integrates some object oriented procedural language with a logical or functional programming language. The similar connection is also presented in this thesis, but at more general level.

The Oz[Muller 95] programming language is very popular, for example. The Oz advantage is in implementing the inference mechanism into the computing, so the programs can be made in combination of deterministic and nondeterministic manner. The OLI language (Objects and Logic Integration) [Lee I., Lee II.] is another representative of this branch (OLI was derived from Smalltalk).

3.2 Processes and knowledge – a heterogeneous example

I would like to mention a paper [Hrubý MOS01], where the first ideas of my work have been published. The paper starts with the SIMLIB/C++ simulation library [SIMLIB HP] and a logic programming language called (SWI-)Prolog [SWI-Prolog HP] oftenly used as a knowledge modelling language[2]. I described a way of interconnecting those two tools, the principle and reason of interconnecting two very different modelling approaches. Result has been shown on a simple example of city traffic.

The crossings - model

Our task would be to model a city described with a net of roads and crossings. We also need to model some cars traveling from some A point to some B point. We are getting a bi-directional graph and path-searching task.

That problem contains two components: a knowledge component and a process part. The graph itself is the knowledge part – it represents a database of roads and their crossings or a distance between two crossings. Car ride is a process-oriented part. Obviously, it is possible to model the whole problem in a single tool, like by the SIMLIB/C++ library (or any equivalent language). I would like to show that:

1. The Prolog language is ideal for database (knowledge) operations.
2. Prolog is also ideal for coding the graph searching algorithm.
3. The SIMLIB library is suitable for behavioral description of the car process and to control the whole simulation.
4. It is trivial to complete the model when having an existing tool made of Prolog and SIMLIB.

The knowledge part can be expressed like this (reduced):

```
...
findroad(From, From, []).
findroad(From, To, W) :-
       find(From, To, Next),
       findroad(Next, To, NN),
       append([Next], NN, W).
```

And the process part in SIMLIB (reduced):

```
class Car : public Process {
       ...
       void Behavior(){
          double dist;
          term_t next;
          while (Act != To) {
            // calling Prolog
```

[2] In [Madsen 00], there is a note about the Primula language, made by connecting Prolog and Simula 67.
Salvesen, A. Wang, A.: Primula - An Integration of Prolog and Simula. Report no. 781, Norwegian Computing Centre, 1986.

```
                findinc(Act, To, &next, &dist);
                Wait(dist/50);
                Act=next; // Arrival to next
            }
        }
};
```

Road (long 200 meters) from an A cross to a B cross is inserted by `road(a,b,200)` predicate. This way of modelling is rapid, comfortable and flexible. We may work in three modes of interconnecting the Prolog and C++ (SIMLIB):

1. The simulation is driven by C++, Prolog is called to decide problems based on its knowledge database.
2. Simulation control is in Prolog competence. Prolog calls SIMLIB for practical actions. Profit is in the Prolog inference mechanism which may help as a task solver (like in the AI) in our simulations.
3. The simulation flow (steps in the simulation experiments) freely moves among the both languages. This way of simulation control is probably the most useful and should be the target of heterogeneous simulation tools development. But there are some technical problems in implementation of such a tool.

3.3 Design of heterogeneous models

Motivation for the heterogeneous modelling arises from the previous text. Let us comment two approaches to achieve the heterogeneity:

1. **Uniform platform for sub-system interconnection** – that is being solved in this work. Unified principle of inter-system communication is a big advantage of this approach as well as a formal base of architecture of the resulting system. Strict adaptation of sub-systems may be limiting in some applications.
2. **Spontaneous approach** – every interconnection of two sub-systems is different. There is no unified communication platform. The communication is limited to sub-systems having their created interfaces. We can adapt the communication to special characteristic features of the sub-systems and possible implementation may be (or seams to be) very fast. Fragmented and incompatible communication is a certain disadvantage of this solution.

3.4 Object oriented technology (some notes)

My work arises from Smalltalk [Goldberg 84, Goldberg 89, Squeak HP], Self [Ungar 91, SELF HP] and Beta [Beta Mjolner]languages. Smalltalk started the object orientation with persistence, interactivity and reflection. Those terms are very important for my thesis and so I would like to explain them briefly.

Interactivity

Languages are different in style how users make their applications. We know the style practiced in C++, Pascal and so on. But there is also a style called *interactive*. The common style cycles usually in these steps:

1. Creating a program (text editing).
2. Program compilation, correcting some syntactical errors announced by the compiler.
3. Application run. We can watch some logic errors in the application design.
4. Results analysis (may be damaged by a logic error in program).

We separate a programming phase, running phase and analysis phase and it is not very comfortable as probably everyone would agree. So-called interactive environment (Smalltalk, Prolog) allows to program an application during its run or to test just a single segment of its code. All program dynamic modifications are allowed, including all changes in structure of classes.

Persistence in Smalltalk (and Self)

Smalltalk developers emphasize open systems, re-usability of code and a feature called persistence – system objects are saved in a persistent (permanent) memory device (hardisk) and so Smalltalk works as an object oriented database system. Conception of simulation models developed in Smalltalk-like systems can be very interesting. We can create our models in a prototype interactive manner with a cooperating gradual simulation of them.

Let us start with imagination of persistent interactive non-class OO environment – like Self is. A simulation model is a net of interconnected objects (we work only with persistent objects in Self-like systems). Net of objects is an identical copy of its previous text program (compilation is invertible). Every step of simulation experiment gives a fundamentally equal simulation model (step of simulation transforms A model to B model which is an object net as well).

If we understand the text program to be a simulation model and the Self-like system is able to export its model to the text program again, then every step of simulation (except the model time) makes a simulation model. We can develop (and debug) our models by simulation.

Chapter 4

Approach

There are some opinions, motivations and partial solution of heterogeneous models mentioned in this thesis. The unified communication platform seams to be the most useful way of constructing these heterogeneous systems and this approach hes been chosen to be a formal background for my whole work.

The unified communication and control platform is supposed to provide interconnection of sub-systems at levels:

1. *Methodologic* – searching for some common attributes of totally different computing paradigms.
2. *Realizing* – design of sub-systems, style of communication and controlling the sub-systems.
3. *Physical (implementation)* – inter-sub-system communication on a layer of data transfers and remote procedure calls.

Solving the problems specified in the goals of the thesis should lead to a complete and compact methodology of preparing systems to the heterogeneous analysing, decomposing, implementation and completing. The first task will be in some mathematical research trying to find that common attributes in all sub-systems which can be used as a communication interface. The following list shows my approach to fulfil my goals, it means to make an *environment for modelling of heterogeneous systems*:

1. Creating a general interconnection platform (AIN) as a formal base for making heterogeneous models (details in 5).
2. Methodology of making the heterogeneous models using the AIN platform (details in 5.2).
3. Analysing the real-world objects and designing a method structuring systems to sub-models with an accent to the highest abstraction, efficiency and *re-usability* of models. This method is a methodologic recommendation of building heterogeneous models from the partial sub-systems (details in chapter 5 of the thesis).
4. Design of a new simulation language HELEF which will practicaly implement ideas of the heterogeneous modelling in the AIN style (details in 5.3).
5. Showing the range of AIN applications, examples and tutorials (details in 5.4).

Chapter 5

The results

5.1 Automatic Information Net (AIN)

The AIN method (Automatic Information Net) introduces a slightly different programming approach and semantics – a program is a set of declarations. Every declaration represents a relation between a identifier (variable) and some expression. The internal control AIN mechanism has to keep data consistence of all declarations in the whole program (model).

We will work with an object *term* often ly in this chapter. Let us say, that from the AIN point of view, an o object is a *data element* containing some $value(o)$ attribute carrying its *current value* – object is a memory element of the system. For example, $a + b$ expression is an object having some value. When talking about the values – in the programming languages, we use variables to keep some values. Let us demonstrate a basic difference between AIN and a general purpose programming language:

- In the common programming language, getting a value of some expression means to start executing that expression (calling a function, assignment to a variable).
- Int the AIN method, query to some o object's value returns a contents of $value(o)$. $value(o)$ is enumered in moment when AIN detects some conditions predicting a possible change of o object's value. In case of $a + b$ object, it is a situation when a or b changed its value.

Declarations reference each other. Declarations and their mutual references make *a net*. If an A declaration uses B declaration to derive its value, the net contains an edge from A to B.

Decomposing the data $\times$ processes

The AIN method has arised from integrating the declarative and imperative ways of computing. For this reason, the AIN method includes also some sequential computings together with the net of linked expressions. The sequential parts (called the actions or processes) are in form of statement sequences which can modify context and structure of the net in every step of its evolution. So, the AIN net contains some data and some process objects – both treated in the same way.

The following definitions are based on so-called – *a simulation object* term. Simulation object is an object which is a data object or an action object. Let us establish some *universum* $\mathcal{U}$ covering values of all primitive data types - for example, all numbers and an empty value called ϵ.

Definition 1. *We will call so element to be a simulation object if it contains (inner) attributes:*

- Id – *object identity,* $id \in \mathbb{N}$.
- Value – *value of the object,* $value(so) \in \mathcal{U}$.
- Type – *enum type* $\{data, process\}$ *to determine data and process simulation object.*

- Contents – *some function to derive value of the object. Specified very generally.*

We denote $value(so)$, $type(so)$ and *contents(so)* to access attributes of so object. Every object has got its identity (attribute id). Object and its id will be associated in the following text. Two objects o_1 and o_2, having $id(o_1) = id(o_2)$ are understood to be identical (including all their attributes) and we type $o_1 \approx o_2$. We will work with sets of simulation objects. An actual set of all simulation objects included in the system will be marked OS (Objects in System).

AIN - basic definitions

I am going to describe the AIN method in these levels of abstraction[1]:

1. *Net of the data objects* – definition of data simulation object and its relations to other objects. Set of objects and their relations makes a structure of the AIN net.
2. *Net evolution, context of the evolution* – principle of enumerating the object's value in context of the AIN net.
3. *The process object and context of the process part* – definition of the process simulation objects and controlling their activity.
4. *The complete context (data, processes)* – definition of the complete AIN evolution context covering enumeration of data objects and the process control.
5. *Run and termination of the computation* – evolution of the AIN net system from the starting context towards the terminating context.

Data simulation objects

Let us have an actual set OS of simulation objects. We can describe each simulation object $so_d \in OS$, $type(so_d) = data$ by expression

$$so_d = expression(so_{d_1}, ..., so_{d_{dmax}});\ dmax \in \mathbb{N} \tag{5.1}$$

where $\{so_{d_1}, ..., so_{d_{dmax}}\} \subseteq OS$ are simulation objects too and so_d uses them to derive its value ($contents(so_d) = expression(so_{d_1}, ..., so_{d_{dmax}})$). This set $\{so_{d_1}, ..., so_{d_{dmax}}\}$ will be called *object so_d base* – $Base(so_d)$.

$$Base(so_d) = \{so_{d_1}, ..., so_{d_{dmax}}\} \subseteq OS \tag{5.2}$$

We say that so_d is dependent on objects in $Base(so_d)$. To denote relations between $so_d \in OS$ and $Base(so_d)$ we establish:

$$Relations(so_d) = \{(b, so_d) | b \in Base(so_d)\} \subseteq OS \times OS \tag{5.3}$$

All data relations of all $o \in OS$ (except the relations to process objects) are:

$$R = \bigcup_{o \in OS} Relations(o) \tag{5.4}$$

Example 1. *Let us mention some examples of the data simulation objects:* $OS = \{1, (1+2), a, (a+b)\}$*. The object* $a+b$ *has got these attributes:* $type(a+b) = data$*,* $contents(a+b) = a+b$*; and* $value(a+b)$ *is not known yet (*$= \epsilon$*).*

Let us start with definition of the AIN-net made of data and process simulation objects and their mutual relations:

[1] Full description is available in the thesis. This is only a reduced version.

Definition 2. *The AIN-net is a tuple* $S = [OS, R] = [OS \cup \{AINnoevent\},\ R]$*, where*

1. *OS is a set of simulation objects. OS contains a special element $AINnoevent$ which will be described in the following text.*

2. *R is a dependence relation on simulation objects $R \subseteq OS \times (OS \setminus \{AINnoevent\})$.*

AIN-net evolution

Let us operate the $AIN = [OS, R]$ net as an AIN-system $\Sigma = (OS, R)$ having OS and R attributes. Net *AIN* expresses a static structure of the model, *system AIN* expresses a dynamic state of the net (values of object's attribute). Some more attributes will be added in the next chapters as the AIN-system definition will extend.

Definition 3. *Let $AIN = [OS, R]$ is an AIN-net. Computing AIN-system $\Sigma = (OS, R)$ represents a computing state of AIN.*

Turning a state of AIN-system $\Sigma = (OS, R)$ onto $\Sigma_2 = (OS_2, R_2)$ we mean a occurrence of one of the following conditions:

- Changing of some (inner) attribute of $o \in OS$ simulation object to $o_2 \in OS_2$ so that $o \approx o_2$.

- Changing of identity numbers in OS so that $\exists o \in OS : \nexists o_2 \in OS_2, o_2 \approx o$ or vice-versa.

- Changing the dependence relation R to R_2 that $R \neq R_2$. Relations (a, b) a (a_2, b_2) where $a \approx a_2$ and $b \approx b_2$ are understood to be identical.

We establish a function $chValue$ which converts value (attribute $value$) of $o \in OS$ in $\Sigma = (OS, R)$"

$$chValue : \ (\Sigma, o, val) \mapsto \Sigma_2, val \in \mathcal{U} \tag{5.5}$$

so that $\Sigma_2 = (OS \setminus \{o\} \cup \{o_2\}, R)$ contains $o_2 \in OS_2$:

1. $o_2 \approx o$.

2. $value(o_2) = val$.

Context of AIN-system is a continuous turning between two states described with two sets $Marked$ a $Changed$ where:

- $Marked$ denotes a set of objects whose base (values $value$ of base-objects) was modified in previous step and so these objects in $Marked$ must be re-evaluated.

- $Changed$ denotes a set of objects which changed their $value$ during the re-evaluation (and will cause a re-evaluation of on them dependent objects $\{o \bullet | o \in Changed\}$).

Let us define a *AIN-system context*:

Definition 4. *Let us have an AIN-net $AIN = [OS, R]$. 3-tuple $AC = (\Sigma, Marked, Changed)$ is called a context (evolutionary state) of the AIN-system, if:*

1. *$\Sigma = (OS, R)$ is an AIN-system.*

2. *$Marked \subseteq OS$, $Changed \subseteq OS$.*

Definition 5. *AIN-context $AC = (\Sigma, Marked, Changed)$ is called* valid *if:*

1. *$Marked = \emptyset \wedge Changed \neq \emptyset$ or*

2. $Marked \neq \emptyset \wedge Changed = \emptyset$ *or*

3. $Marked = Changed = \emptyset$.

Valid AIN-context AC is called:

- *Stabilized*, if $Marked = Changed = \emptyset$ (AC_U).
- *Marked*, if $Marked \neq \emptyset \wedge Changed = \emptyset$.
- *Changed*, if $Marked = \emptyset \wedge Changed \neq \emptyset$.

Performing the evolution steps from a AC_1 state to another AC_2 state is the operation principle of the AIN-system. Let us define a single evolution step of an AIN-system.

Definition 6. *A valid AIN-context $AC_1 = (\Sigma_1, M_1, CH_1)$ derives a valid context $AC_2 = (\Sigma_2, M_2, CH_2)$ if:*

1. *AC_1 is stabilized, $M_2 = AINnoevent\bullet$, $CH_2 = \emptyset$ and $\Sigma_2 = \Sigma_1$,*
2. *AC_1 is changed, $M_2 = \bigcup_{i \in CH_1} i\bullet$, $CH_2 = \emptyset$ and $\Sigma_2 = \Sigma_1$,*
3. *AC_1 is marked and M_2, CH_2, Σ_2 attributes we get by:*
   - $M_2 := \emptyset$, $\Sigma := \Sigma_1$,$CH_2 := \emptyset$
   - $while(M_1 \neq \emptyset)\, begin$
   - m :=*take a element of* M_1
   - $(flag, \Sigma) := reeval(\Sigma, m)$
   - $if\,(flag = TRUE)\, then\, CH_2 := CH_2 \cup \{m\}$
   - end
   - $\Sigma_2 := \Sigma$

We type $AC_1 \vdash AC_2$ to denote rewritting AC_1 to AC_2. The$\vdash$ operation defines a single evolution step of the net. Only valid contexts are present in the AIN evolution, so we will understand *context* to be a *valid context* in the following text.

Reeval: $(\Sigma, so) \mapsto (Boolean, \Sigma_2), so \in \Sigma$

It has been presented here many times before: $value(so)$ is the basic attribute of the so simulation object. Every simulation object contains also some $contents(so)$ procedure which describes how to get a current value of the object in context of the whole system.

The $reeval$ function returns a new Σ_2 AIN- system for a given Σ AIN-system and a data objects so. Resulting Σ_2 system contains a re-evaluated object $so_2 \approx so$ and $reeval$ also returns a flag of modification:

$$reeval : (\Sigma, so) \mapsto (Boolean, \Sigma_2) \tag{5.6}$$

where $so \in OS$ and $\Sigma = (OS, R)$, $\Sigma_2 = (OS_2, R)$ are AIN-systems.

5.1.1 AIN-net evolution sequence

The AIN-evolution is here a way of doing computations. It has its start in so-called starting context, its running and a terminating state can be expressed as well.

Definition 7. *Starting context of the* $\Sigma = (OS, R)$ *AIN-system is a* $AC_0 = (\Sigma, M_0, \emptyset)$ *so that:*

1. $M_0 = \{o \in OS | \bullet o = \emptyset\} \setminus \{AINnoevent\}$.
2. $\forall o \in OS : value(o) = \epsilon$.

Definition 8. *AIN-evolution of* $\Sigma = (OS, R)$ *is a sequence of contexts* $AC_0, AC_1, AC_2, ...$*, where:*

1. $AC_0 = (\Sigma, M_0, \emptyset)$ *is a starting marked state of the AIN-system.*
2. $AC_i \vdash AC_{i+1}$ *for all* $i = 0, 1, 2, ...$*;* $i \in \mathbb{N}$.

The evolution does not specify a default terminating state. With no included processes we can express a terminating state as a stabilized context $AC_F = (\Sigma_F, \emptyset, \emptyset)$ that:

$$AC_F \vdash (\Sigma_{F1}, AINnoevent\bullet, \emptyset) \vdash (\Sigma_{F2}, M_x, \emptyset) \vdash (\Sigma_{FF}, \emptyset, \emptyset) = AC_{FF} \tag{5.7}$$

Example 2. *Let us have a net* $S = [OS, R]$*, where* $OS = \{1, b, b < 3, b + 1\}$ *and* $R = \{(1, b), (b < 3, b), (b, b < 3)\}$*. The starting context is* $AC_{start} = (\Sigma_{start} = (OS, R), M_{start}, \emptyset)$ *and* $M_{start} = \{1\}$*. The evolution will proceed as follows:*

$$(\Sigma_1, M_{start}, \emptyset) \vdash (\Sigma_2, \emptyset, \{1\}) \vdash (\Sigma_3, \{b\}, \emptyset) \vdash (\Sigma_5, \emptyset, \{b\}) \vdash (\Sigma_6, \{b < 3\}, \emptyset) \vdash$$
$$(\Sigma_7, \emptyset, \{b < 3\}) \vdash (\Sigma_8, \{b\}, \emptyset) \vdash (\Sigma_9, \emptyset, \{b\}) \vdash (\Sigma_{10}, \{b < 3\}, \emptyset) \vdash (\Sigma_{11}, \emptyset, \emptyset) \vdash$$
$$(\Sigma_{12}, \emptyset, \emptyset) = AC_{NatTerminate}$$

5.1.2 The process part

We are going to include a process-oriented part of the AIN method now. Processes are the simulation object linked to AIN-net which are activated in exactly specified moments. The $\vdash$ evolution is suspended during their operation. Processes allows to formulate sequentially interpreted parts of models and they also allows to do some structural inferences in the AIN-net (adding a new object, removing and moving objects).

Definition 9. *Process* $P = (Body, StartC, ContinueC)$ *is a simulation objects containing attributes:*

- $Body$ *– body of the process made of statements sequence.* $Body = (c_1, c_2, ..., c_{pc})$.
- $StartC$ *– data simulation object specifying a process start condition*
  $Value(StartC) \in \{TRUE, FALSE, \epsilon\}$.
- $ContinueC$ *– data simulation object specifying a run condition (a process runs while* $Continue(P) = TRUE$*)*
  $Value(ContinueC) \in \{TRUE, FALSE, \epsilon\}$.

Objects $StartC$ and $ContinueC$ may be empty ($= \epsilon$). In $[OS, R]$ with process $P \in OS$ also $StartC \in OS$ and $ContinueC \in OS$. Let us add a formula for set of relations of a process simulation object $P \in OS$:

$$Relations(P) = \{(StartC, P), (ContinueC, P)\} \setminus \{(\epsilon, P)\} \tag{5.8}$$

A process p starts when $StartC(p)$ turns on true. It means, in sequence:

$$\Sigma = (\{p, StartC(p), ...\}, \{(StartC(p), p), ...\} \tag{5.9}$$

$(\Sigma_1, \{StartC(p), ...\}, \emptyset)\vdash (\Sigma_2, \emptyset, \{StartC(p), ...\})\vdash (\Sigma_3, \{p, ...\}, \emptyset)$. Process management is driven by a higher layer called Complete Simulation Context.

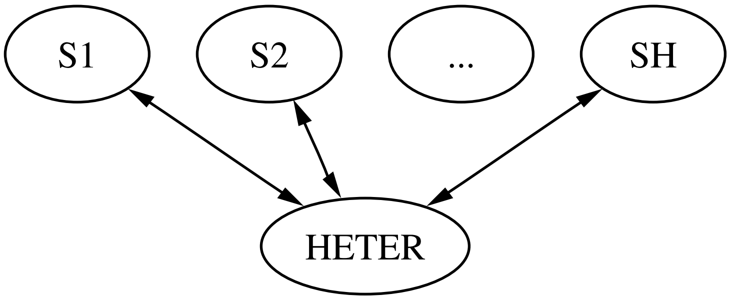

Figure 5.1: Heterogeneous system

5.2 A heterogeneous system made in the AIN

We have defined an universal computing system called AIN in the previous text, which is based on the AIN communication net. Such a net $S_1 = [OS, R_1]$ can be connected to another net $S_2 = [OS, R_2]$ so that a single whole is created. But $S = [OS_1 \cup OS_2, R_1 \cup R_2]$ only if S_1 and S_2 are isolated subnets.

Our goal is in building of connected systems made of communicating sub-systems. Such sub-systems $[OS_s, R_s]$ contains objects like $obj \in OS_s$, $\exists o \in Base(obj) : o \notin OS_s$ – object obj is dependent on another system's object.

Let us have a heterogeneous system:

$$\{S_1, S_2, ..., S_H, HETER\} \tag{5.10}$$

integrating various sub-systems $S_1, ..., S_H$ and a main control sub-system $HETER$ (figure 5.1).

We would like to emphasize, that technical support of inter-sub-system communication is not the goal of this method. The reason is only in the general platform for connecting different computing paradigms. sub-system interface will be defined as a set of messages necessary to understand for a sub-system to be plugged.

5.2.1 Initialization of a heterogeneous system

Let every sub-system $S_i, i = 1..H$ contains some *local* and *global* objects (data and process simulation objects). Only the global objects $S_i globals = \{o_{i,1}, ..., o_{i,max_i}\}$ are included to the heterogeneous interconnection.

The $HETER$ system broadcasts a message $S_i.getYourObjects()$ to each sub-system. A sub-system is expected to return its $S_i globals$ set. Complete set of all objects is:

$$allObjects = \bigcup_{i=1..H} S_i globals \tag{5.11}$$

Sub-systems must receive the following messages operating their global objects. This package of messages creates this heterogeneous communication/control interface:

- *GetValue(o)* – return attribute $value$ of the o object.
- *SetValue(o,val)* – set the $value$ attribute $value$ of o,$val \in \mathcal{U}$.
- *Name(o)* – return a name of o.
- *RequiredNames(o)* – return $\{name(obj)|obj \in Base(o)\}$ containing names of $Base(o)$ objects.
- *Reeval(o)* – proceed $reeval$ in o. Return $\{TRUE, FALSE\}$ flag.
- *RunUntilChanged(op)* – interpret an op process until its interrupt or termination. It returns an appropriate *changes report*.

The control $HETER$ system collects all global objects of all sub-systems and creates a set of the simulation objects $OS = allObjects$. We will handle the simulation objects with their names in this phase. Names are used for exact identification of the objects in space of the heterogeneous system. Let us suppose just a single dimensional (linear) namespace for new.

Every object has got its name and requires a set of objects (names) specified by its base which is given by call $requiredNames(o)$. The $HETER$ system has to find the required objects in the second phase of this initialization. Let us define a function called $findObject(name)$ returning a reference to an object having name $oName$:

$$findObject(oName) = \begin{cases} object & object \in allObjects, name(object) = oName \\ \epsilon & otherwise \end{cases} \tag{5.12}$$

Object having name $oName$ need not to be included in $allObjects$ – then the heterogeneous system will not work as a whole. Let us suppose the complete system. Let the function $findObjects$ is $findObjects(names) = \{findObject(n)|n \in names\}$.

The relation set R for set of objects $OS = allObjects$ is created from:

$$relationsOf(o) = \{(x, o)|x \in findObjects(requiredNames(o))\} \tag{5.13}$$

$$R = \bigcup_{o \in OS} relationsOf(o) \tag{5.14}$$

So, the system $[OS, R]$ of all heterogeneous distributed objects is completed.

The starting CSC-context is again $CSC_S = (q_S, ((OS, R, <>, <>), M_{Start}, \emptyset), \epsilon)$. Simulation of the heterogeneous model start with $CSC_S \vdash_C CSC_{S+1}$ and may terminate at $CSC_{NatTermination}$ state or $CSC_{until(name)}$ state with an ending condition $name$.

5.3 The HELEF Language

The HELEF language [Hrubý MOS02, Hrubý MOS03, Hrubý ASU02] has been designed to transparently integrate the declarative and imperative programming. During its development, there were two new modelling methodologies invented:

- General heterogeneous modelling (the AIN method).
- Real modelling of the real-world objects (RWRM).

The AIN method defines a concept of declaration driven programming in the computer modelling and the real-world modelling method extends this concept with a methodology of modelling the structure and behavior of real world. Moreover, HELEF also implements interactivity, persistence and reflectivity.

Is summary, the HELEF language combines several concepts of modelling and programming:

- Declarative and imperative programming in one language.
- Heterogeneous modelling (multi-paradigm programming) based on the AIN method.
- Non-class object oriented modelling[Ungar 91].
- Interactive creating of models.
- Persistence of models.
- Distributivity.
- Inner simulations, nondeterministic computing and task solving.

The HELEF language represents a heterogeneous interconnection language in meaning of theory of heterogeneous modelling. Most of the HELEF's semantic has been defined in the previous text.

5.3.1 Simulation objects

There are various types of simulation objects in the HELEF language. Types of HELEF specific simulation objects are:

- Expression – a mathematical expression. It works as a right side of the assignments and a starting condition in declarations and processes (StartC, if, while).
- Information – a basic declaration. Defines a relation between some object and its name.
- Name – a system object which is generated automatically out of the declaration.
- Structure – a list of objects. Defines a namespace.
- Process – a sequential code.

5.3.2 Structures

There are three types of structures in the HELEF language specification. They represents three aspects of modelling of the reality and real-world objects as was deeply described in the fifth chapter of the thesis. That theory, which is the second contribution of my thesis, will be mentioned here only briefly.

The reality has a structure and behavior. We recognize active and passive objects depending on their behavior and we use processes to code that behavior as a sequence of actions. When thinking about the world's structure, we find that there is something like a hierarchy. The basic element of this hierarchy is called an *environment* here.

An environment is a physical or abstract surrounding of objects. It is a bordered space declaring an existence or non-existence of some conditions. Objects included in the environment are called *entities*. There may be also some inner environments included and that models the mentioned hierarchy of the world.

In the HELEF specification, there are these types of structures:

- Environments – structures which provides set of conditions for the included entities.
- Entities – active or passive objects which are dynamic parts of the world hierarchy. They can migrate.
- Properties – abstract objects keeping some information. They are some knowledge for environments and entities to share.

A structure is:

```
type [["@"] inhName] "{"
{ locals; exports; }
{ items }
"}"
```

where $type \in \{environment, entity, property\}$ and $inhName$ is a name of another structure to inherit from.

```
world = environment {
  time := 0
  time(AINnoevent) = time + 0.01
  process(time=1) {
  }
}
```

5.3.3 Processes

In the AIN definition, process is an object having some sequential code. Process is started in a moment when $StartC$ condition becomes true and it runs until it modifies some global object or $ContinueC$ is false. $StartC$ and $ContinueC$ are internal attributes of a process.

```
process [["@"] inhName] "(" [condition] ")" "{"
  { "local" {id} ";"}
  { command }
"}"
```

Syntax of the process internal commands is similar to other languages like C and Pascal. Process header declares a process to inherit from (Beta, Simula semantics with "inner") and a $StartC$ condition. We may define process with or without starting condition.

5.3.4 Compilation and run

The HELEF language is now being implemented in the Smalltalk language because Smalltalk is very close to HELEF in philosophy, style and object orientation. Every simulation object can exist independently, is perzistant and may be interactively inserted to another simulation object. Non-class OO differs a lot because there is no static and dynamic form of object – every object in text form can be loaded to the system, processed and exported back to the text form.

```
t := 'time(AINnoevent)=time+1' helef
```

Let us type this code in the Smalltalk workspace and press "do it". Certain tree of simulation objects is created and we can manipulate it as a standard Smalltalk object referenced with t.

```
model := 'environment {
   process(time=1) {}
 }' helef becomeModel
```

This statement will create an environment structure which will provide a basic environment of the whole model. Objects referenced as t and $model$ are still independent. Following statement will insert t into $model$:

```
model addObject: t
```

5.4 Modelling in the HELEF language

Model time

Every simulation model in the HELEF language is an *environment* structure and is called *world* by the HELEF convention.

```
world = environment { // environment of the simulation model
}
```

Model time is not defined by default. The time axis can be specified by the *basicTimeProperty* for example.

```
basicTimeProperty = property {
    local timestep; //
    time := 0
    time(AINnoevent) = time + timestep
}
```

and specialized when inserted into a model of world:

```
world = environment {
    property basicTimeProperty {
       timestep := 0.01 // insertion of time
    }
}
```

Event driven simulation

An event is defined by a process with a starting time condition $StartC$. Let us have a set of blocks $\{eventA$, $eventB, ...\}$ for serving the events in form of processes.

```
eventA = process { // proces called eventA represents
                   // servicing of eventA
}
```

and let `eventA` is scheduled at $time = 0$.

```
world = environment { // we insert a process
  // time = 0
  process eventA {
     StartC = (time=0);
  }
}
```

Event block $eventA$ is started when $StartC$ becomes valid (the AIN simulation engine puts this block into the ready processes queue). It proceeds its activity – schedules some future events and it can terminate itself or re-schedule itself by setting $StartC$ at next time moment or $ContinueC$ to continue. Scheduling a future event $eventX$ can be declared like this:

```
eventX.StartC = (facility.Free=true); // active the event when
   // the facility becomes available
eventX.StartC = (time>=at); // start with a time condition
```

Continuous simulation

The AIN net and its computing evolution is a direct model of continuous systems diagrams with integrators. An integrating block is (very simplified):

```
integrator = entity {
  local input;
  output(time.event) = output+timestep*input
  output := 0
}
```

An example with the integrator object:

```
block = environment {
  i = entity @integrator {
    output := 1
    input = x + (-0.3*output)
  }
  x := 0
  x(time=1) = 1
}
```

`Block` specializes the general pattern integrator – it creates the i object with particular input and output objects. The x input has got zero value till $time = 1$ when it turns on 1.

Combined simulation

Let us extend the previous example – we add a process with a starting condition:

```
process(i.output>=0.8) {
  x := 0;
}
```

When added to the `block` environment, we define a state condition for a discrete event and we get a combined simulation.

Chapter 6

Conclusions

My thesis comes from a wide study of features in various programming and simulation object oriented (OO) languages and it continuous with a methodology design and program support in creating of the heterogeneous models. Description of some OO languages, multi-paradigm languages and their properties summarizes the current state of research in modelling methods. Smalltalk language has influenced me a lot because of its interactivity, persistence, reflectivity and pure object orientation.

The heterogeneous modelling is probably the future trend in development of modelling methods. Moreover, I suppose that it will find some applying also in programming of larger program products like operating systems, information systems, artificial intelligency systems and other computer applications in the human life. Heterogeneous modelling is oftenly more flexible, comfortable and efficient then the single-paradigm modelling. We only have to develop some methods to decompose the modelled problems and to put that elements back together. Problem of effective interconnecting inside the heterogeneous models is the only barrier in their practical realization. In my thesis, I designed a method of decomposing the modelled system to a heterogeneous system processable by AIN. The AIN net allows most of the sub-systems to communicate and to control their cooperative simulation. Created universal interconnection platform AIN is a main original contribution of this work. The reader can find a compact methodology of adapting various autonomous systems for the AIN method here. For example, integrating the process and event driven simulation languages, continuous simulators, finite state automatons, Petri nets, logic programming languages and other sub-systems covering inputs and outputs of the systems, statistics and visualizations.

Structural and behavioral analysis of the real world has been a second target of the thesis. Attempts for a highly general models allowing their re-usability were the motivation for this part. I designed a method of real-world natural modelling where the problem is formed to hierarchy of environments and movable entities. When some entity is inserted to an environment, both sides gain a set of temporary or persistent derived attributes. Idea of this method was to reduce a modelled object to only those attributes which the object would contain without interaction to its surrounding. Reduced objects are more general and easier to re-use in other models. This methods makes a theoretical background for realization of an interactive and persistent simulation tool which can be equipped with a extensible library of re-usable simulation models. Such an environment would be a significant benefit for making the simulation modelling more effective.

Theoretical results of the AIN method and method of natural modelling of real world have led to a design of a brand new heterogeneous simulation language called HELEF. The HELEF language has been described very briefly because of available size of this text. The accent was put on describing the relations between chapters 4 and 5 which HELEF presents in more practical meaning. The language specification of HELEF is still in development, but only some small details are being finished. Currently, the HELEF language is in process of implementation in Squeak Smalltalk.

I assume some further development of the HELEF language – finishing of its specification, creating a working implementation of its simulation kernel and user interface and basics of the mentioned library of simulation abstractions.

Bibliography

[Beta Mjolner] Beta Language Introduction, Mjolner Informatics Report, MIA 94-26, September 2001

[Blazevic 01] Blazevic, M., Budimac, Z.: An Environment-based Multiparadigm Language, In: Proceedings of the 1st Workshop on Multiparadigm Programming with Object Oriented Languages, Budapest, Hungary, 2001, ISBN 3-00-007968-8

[Buss 98] Buss, A., Jackson, L.: Distributed Simulation Modeling: A Comparison of HLA, CORBA, and RMI, In: Proceedings of 1998 Winter Simulation Conference, 1998

[DEVS] Conception, A. I., Zeigler, B. P.: DEVS Formalism: A Framework for Hierarchical Model Development, IEEE Trans. on Software Engineering, vol. 14 no. 2, Feb 1988

[Fishwick TR] Fishwick, P., Zeigler, B. P.: A Multimodel Methodology for Qualitative Model Engineering, technical report

[Goldberg 84] Goldberg, A.: Smalltalk-80: The Interactive Programming Environment, Addison-Wesley series in Computer Science, 1984, Xerox, ISBN 0-21-11372-4

[Goldberg 89] Goldberg, A., Robson, D.: Smalltalk-80: The Language, Addison-Wesley series in Computer Science, 1989, ParcPlace Systems, ISBN 0-20113688-0

[Chambers 92] Chambers, C.: The Design and Implementation of the SELF compiler, an Optimizing Compiler for Object Oriented Languages, phd thesis, Stanford University, 1992

[Janoušek 02] Janoušek, V., Kočí, R.: PNtalk - An Open System for Prototyping and Simulation, In: Proceedings of The 28th ASU Conference, Brno, CZ, FIT VUT, 2002, p. 133-146, ISSN 1102-593X

[Lee I.] Lee, J. H. M., Pun, P. K. C.: On Overview of the OLI Multiparadigm Language and Its Semantics, technical report, The Chinese University of Hong Kong

[Lee II.] Lee, J. H. M., Pun, P. K. C.: Object Logic Integration: a Multiparadigm Design Methodology and a Programming Language, technical report, The Chinese University of Hong Kong

[Liu 96] Liu, Ch.: Smalltalk, Objects, and Design, iUniverse.com, USA, reprint, 1996, ISBN 1-58348-490-6

[Madsen 00] Madsen, O. L.: Towards a Unified Programming Language, ECOOP 2000, LNCS 1850, pp.1-26, Springer-Verlag Berlin Heidelberg 2000

[Muller 95] Muller, M., Muller, T., Roy, P. V.: Multiparadigm Programming in Oz, výzkumná zpráva RR 95-16, In: “Workshop Visions for the Future of Logic Programming: Laying the Foundations for a Modern Successor to Prolog”, Portland, Oregon, 1995

[Ören 02] Ören, T.: Future of Modelling and Simulation: Some Development Areas, Proceedings of the 2002 Summer Computer Simulation Conference

[Praehofer 96] Praehofer, H.: An Environment for DEVS-Based Multi-Formalism Modeling and Simulation in C++, In: Proceedings of AIS, San Diego, 1996

[Praehofer DEVS] Praehofer, H., Zeigler, B. P.: On the Expressibility of Discrete Event Specified Systems, technical report

[Rumbaugh 91] Rumbaugh, J. and col.: Object-Oriented Modeling and Design, Prentice Hall, 1991, ISBN 0-13-630054-5

[SELF HP] Self HP: http://www.cs.ucsb.edu/labs/oocsb/self/

[Shaalan 95] Shaalan, K. F. S.: Software Development Environment Based on Object-Oriented and Logic Programming Paradigms, phd. thesis, Cairo University, Egypt, 1995

[SIMLIB HP] SIMLIB HP: http://www.fit.vutbr.cz/~peringer/SIMLIB

[Smalltalk HP] Smalltalk HP: http://www.smalltalk.org/

[Squeak HP] Squeak Smalltalk HP: http://www.squeak.org

[SWI-Prolog HP] SWI-Prolog HP: ftp://swi.psy.uva.nl/pub/SWI-Prolog/

[Ungar 91] Ungar, D., Chambers, C., Chang, B., Holzle, U.: Organizing Programs Without Classes, In: LISP and Symbolic Computation: An International Journal, Nederlands, 1991

[Vangheluwe TR] Vangheluwe, H. I., Vansteenkiste, G. C.: A Multi-Paradigm Modelling and Simulation Methodology: Formalisms and Languages, University of Gent, Belgium

[Wadler 92] Wadler,P.: The essence of functional programming, invited talk, In: Proceedings of 19'th Annual Symposium on Principles of Programming Languages, Albuquerque, New Mexico, January 1992

[Zeigler 02] Zeigler, B.P., Praehofer, H., Kim, T.G.: Theory of Modeling and Simulation - Integrating Discrete Event and Continuous Complex Dynamic Systems, reprint, Academic Press, USA, ISBN 0-12-778455-1

Author's bibliography

[Hrubý MOS01] Hrubý, M., Peringer, P., Rábová, Z.: Knowledge-Based Simulator in Object-Oriented Environment, In: Proceedings of the 35th Spring International Conference MOSIS 2001, Hradec nad Moravicí, CZ, MARQ, 2001, p. 75-80, ISBN 80-214-1860-5

[Hrubý MOS02] Hrubý, M., Rábová, Z.: HELEF: A New Simulation Language, In: Proceedings of the 36th Spring International Conference MOSIS 2002, Rožnov pod Radhoštěm, CZ, MARQ, 2002, p. 39-47, ISBN 80-85988-71-2

[Hrubý MOS03] Hrubý, M.: Formal Specification of the HELEF Simulation Language, In: Proceedings of 37th Internation Conference MOSIS'03, Ostrava, CZ, 2003, p. 143-148, ISBN 80-85988-86-0

[Hrubý ASU01] Hrubý, M., Rábová, Z.: Object Oriented Programming Languages in Modelling, In: Proceedings of The 27th ASU Conference, Rattwik, SE, ASU, 2001, p. 55-65, ISSN 1102-593X

[Hrubý ASU02] Hrubý, M., Rábová, Z.: Modelling of Real-world Objects using the HELEF Language, In: ASU Newsletter: a publication of the Association of SIMULA Users., roc. 28, c. 1, Stockholm, SE, p. 47-57, ISSN 1102-593X

[Hrubý 01] Hrubý, M., Rábová, Z.: Processes and Knowledge in One Simulation, In: Proceedings of 7th Conference STUDENT FEI 2001, FEI VUT Brno, Brno, 2001, p. 333-338, ISBN 80-214-1860-5

[Hrubý 02] Hrubý, M., Kočí, R., Peringer, P., Rábová, Z.: Tools for Creating of Multimodels, In: Kybernetes: An International Journal of Systems & Cybernetics, roc. 2002, c. 9, GB, p. 1391-1400, ISSN 0368-492X

[Hrubý 03] Hrubý, M., Kočí, R.: Modern Simulation Techniques and Tools, In: Proceedings of the International Workshop MOSMIC'2003, Zilina, SK, FRI ZU, 2003, p. 7-15, ISBN 80-8070-139-3

[Hrubý MOS04] Hrubý Martin, Peringer Petr, Rábová Zdenka: Modelling of Tamper-Proof Devices, In: Proceedings of 38th International Conference MOSIS'04, Ostrava, CZ, MARQ, 2004, p. 6, ISBN 80-85988-98-4

Author's Curriculum Vitae

 **Ing. Martin Hrubý** (born August 9th, 1976 in Olomouc)
http://www.fit.vutbr.cz/~hrubym

Education:

- 2000 - 2004: PhD study at Faculty of Information Technology, VUT Brno.
- 1994 - 2000: Faculty of Electrical Engineering and Informatics (FEI), VUT Brno. MSc. degree. Diploma thesis referred to modelling and simulation, parallel processing and programming languages.

State Doctoral Exam: June 24, 2002

Research projects

- grant FRVŠ FR0172/2001/G1: Soubor programů pro modelování systémů s aplikací v analýze rizik (a team leader)
- grant GAČR GA102/01/1485: Prostředí pro vývoj, modelování a aplikaci heterogenních systémů (a team member)
- grant FRVŠ FR1831/2002/G1: Moderní metody překladu jazyků (a team leader)
- grant GAČR 102/04/0871: Information system security - research of attacks on tamper-resistant cryptographic hardware (coopearation)
- výzkumný záměr FIT a FEKT: Výzkum informačních a řídících systémů (cooperation)

This thesis is available in the library of the Faculty of Information Technology of Brno University of Technology, and on the web: *www.fit.vutbr.cz/~hrubym*

Abstrakt

Předložená disertační práce je zaměřena do oblasti metod a nástrojů pro výstavbu modelů složitých, rozsáhlých a heterogenních systémů. Výstavba modelů těchto systémů je spojena s hledáním vhodných metod a nástrojů umožňujících návrh a realizaci modelů v různém aplikačním prostředí. Vzhledem k tomu, že metody a nástroje modelování systémů na počítačích již tradičně odrážejí vývoj programování a programovacích jazyků, inspiroval jsem se filozofií a různými modifikacemi objektově orientovaného programování, které jsou obsaženy v několika zajímavých objektově orientovaných jazycích.

Výchozím bodem předložené práce je analýza heterogenních systémů. Modelovaný heterogenní systém rozdělíme na několik sub-systémů, z nichž každý popíšeme a dále řešíme způsobem, jenž je pro daný sub-systém nejvhodnější. Vznikne tak množina sub-modelů, které pak spojíme do celkového heterogenního modelu. Způsob realizace spojení submodelů a zajištění jejich kooperace není triviálním problémem. Můj návrh řešení patří k původním přínosům práce. Jde o způsob propojování sub-modelů založený na komunikační síti AIN (**A**utomatic **I**nformation **N**et), na kterou úzce navazuje simulační jazyk HELEF (**He**terogeneous **L**anguage **E**nvironment, **F**inal version), jehož návrh je rovněž původním přínosem mé práce.