ITHEA

International Journal

INFORMATION TECHNOLOGIES & KNOWLEDGE

2009 Volume 3 Number 2

International Journal
INFORMATION TECHNOLOGIES & KNOWLEDGE
Volume 3 / 2009, Number 2

International Journal "INFORMATION TECHNOLOGIES & KNOWLEDGE" Vol.3, Number 2, 2009

Edited by the Institute of Information Theories and Applications FOI ITHEA®, Bulgaria,
in collaboration with the V.M.Glushkov Institute of Cybernetics of NAS, Ukraine,
and the Institute of Mathematics and Informatics, BAS, Bulgaria.

Publisher: ITHEA®
Sofia, 1000, P.O.B. 775, Bulgaria. www.ithea.org, e-mail: info@foibg.com
Printed in Bulgaria

ISSN 1313-0455 (printed) ISSN 1313-048X (online) ISSN 1313-0501 (CD/DVD)

AN ONTOLOGY-BASED APPROACH TO THE INCOMPLETE SIMULATION MODEL ANALYSIS AND ITS AUTOMATIC COMPLETION

Alexander Mikov, Elena Zamyatina, Evgenii Kubrak

Abstract: *The subsystem of simulation model completion is discussed. This subsystem is one of the components of computer-aided design and simulation system Triad.Net. Triad.Net is dedicated to computer systems design. And it is well-known, that very often investigators do not know about the behavior in details of each component of computer system being under design in the early stages of this process. The paper considers an ontology-based approach for the incomplete simulation model analysis and its automatic completion. A behavior procedure for the undefined element is searched for in special database and included in simulation model. The paper considers the method of model completion, namely, introduces the concept of a "semantic type" and some conditions that should be fulfilled for an appropriate behavior procedure to be chosen. The base ontology of simulation model representation is discussed and the choice of language OWL is explained. The presented example shows the process of simulation model automatic completion, illustrates the use of semantic type and additional conditions. Besides, the paper describes the programming tools which provide an ontology-based automatic completion, presents the semiautomatic completion of partly described simulation model in simulation system Triad and the architecture of the program subsystem being under consideration.*

Keywords: Simulation, ontology, simulation model uncertainty, automatic completion, OWL.

ACM Classification Keywords: I.6 SIMULATION AND MODELING I.6.2 Simulation Languages:J.6 COMPUTER-AIDED ENGINEERING I.2 ARTIFICIAL INTELLIGENCE I.2.5 Programming Languages and Software - Expert system tools and techniques

Introduction

The researchers using simulation as a method of investigations of complex systems often confronted with a problem of analysis of partly described models. Usually the behavior (rules of operation) of some model components is unknown. For example, it is not known, how much time the database search will take and will it be successful (information system is an object of investigations). Another example: a designer of computer networks does not know the exact behavior of router or another device in the early stages of computer networks design. A designer needs only a rough algorithm of data transfer. He doesn't know yet this algorithm in details.

It is clear, that under such conditions simulation process will not provide accurate account of complex system processes. However, despite this fact, a researcher wants to carry out a simulation experiment, and to obtain some results, which should be considered approximate.

The problem is that the simulation system cannot perform a simulation experiment if it lacks even one procedure describing behavior of any element of designed complex system. Therefore substitution of lacking behavior procedures with some "appropriate" ones taken from the standard library is needed to bring the model up to full strength.

This paper describes a process that is known as an automatic completion of a simulation model and considers an ontology based approach towards the problem solving used in simulation system Triad.Net [Mikov, 1995, 2003].

It is necessary to mention, that knowledge-based approach and automation are two factors increasing simulation modeling effectiveness and flexibility.

Related works

Some papers, dedicated to motivations for using ontologies in simulation modeling and role of ontologies in simulation process, appeared last time, for example [Silver, 2006], [Fishwick, 2004, 2005], [Miller, 2005].

So, [Benjamin, 2006] describes an ontology-based solution framework for simulation and modeling and analysis and outlines the benefits of this solution approach.

First of all let us introduce ontologies: «an ontology is an inventory of the kinds of entities that exist in a domain, their salient properties, and the salient relationships that can hold between them», every domain-typically, in this context, some piece of the actual world such as a manufacturing system, an university, a business – has its own ontology, which we refer to simply as a domain ontology [Benjamin,1995]. Ontology represents knowledge in a structured manner, within the structure of a semantic network, consisting of a diagram composed of nodes and arcs. Nodes are dedicated for representation of concepts, but arcs – the relationships among concepts within a particular knowledge domain. So we can see an illustration of simple ontology which relates various Petri nets models together (fig.1.) [Fishwick, 2005].

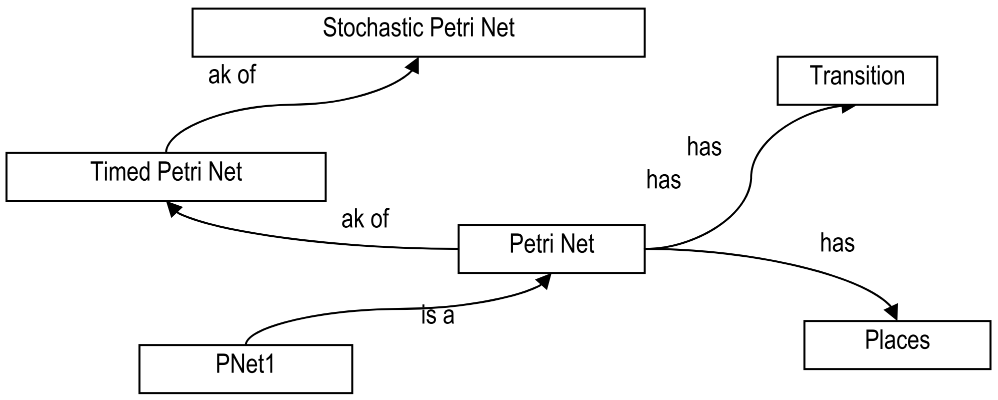


Fig.1.An Ontology for Petri Nets

One can extract such a knowledge from an ontology, represented by fig.1.: (1) A Petri net is a kind of (ak of) Timed Petri Net, which in turn is a kind of Stochastic Petri Net (Timed Petri Nets are types of Stochastic Petri Nets); (2) Each Petri Net is composed of Places and Transitions; and (3) One particular Petri Net is labeled Pnet1 (instance Pnet1 defines one specific Petri Net and may be associated with any business process).

Let us consider the benefits of ontologies. So we can conclude that ontologies may be used to share a common understanding of the structure of information, enable reuse of domain knowledge, make domain assumptions explicit, separate domain knowledge from operational knowledge, and analyze domain. Several of modeling researches may use the same ontology providing links between the concepts and so it increases the potential for interoperability, integration and reuse of simulation artifacts. Ontologies are useful across the simulation modeling and analysis lifecycle, particularly in the problem analysis and conceptual model design phases, they are essential in facilitating simulation model interoperability, composition, and information exchange at the semantic level.

First of all we will show how ontologies may be used in simulation process consisting of several parts: purpose of simulation model defining, acquiring and analysis of system description, determining and classification modeling objectives, determining object roles, boundaries and level of detail, data collection and data analysis, detailed model designing.

So while modeler defines purpose of simulation model design the role of ontologies may be determine as "terminology harmonization to enable shared and clear understanding".

The stage of conceptual model design includes such activities as system description validation, model boundaries and level of modeling abstraction identification, model objects and their roles identification, model structure and logic determination. Ontology knowledge is used here to determine the unambiguously differentiated abstraction levels, to map system objects to model objects and to identify an appropriate model objects role. Moreover ontological analysis helps reason with system constraints to facilitate determination of model logic.

The stage of data analysis suppose the fulfilling such an activities as data reduction, statistical analysis performing, data and text mining. Ontologies play a major role in detailed data analysis.

Next stage of simulation process is detailed model design. It have to provide a refinement of simulate model objects, their validation, moreover, a refinement of model structure and model logic. The primary role of ontologies in detailed model design is in the detailed analysis of information about objects and constraints. This involves mapping the simulation model constraints to specifications of real world constraints that are found within the domain system descriptions.

With increased use of distributed intelligence approaches to simulation modeling (distributed simulation, federated simulation, agent based simulation, etc.), ontologies play a critical role in simulation integration and simulation composability (for example, HLA [Gustavson, 2004], [Rathnam, 2004]). "Composability is the capability to select and assemble simulation components in various combinations into simulation systems to satisfy specific user requirements" [Petty, 2003]. The components to be composed may be drawn from repository. One can distinguish two kinds of composabilities: syntactic and semantic. Syntactic composability deals with the compatibility of implementation details such as parameter passing mechanisms, external data accesses, and timing mechanisms. Semantic composability, on the other hand, deals with the validity and usefulness of composed simulation models [Petty, 2003].

Nowadays simulation models may be composed using components which were developed by different teams in different domains. It is necessary to provide valid interaction between components. One can name components being integrated as federate and simulation model which was developed with these components (federates) is often called a federation. There are multiple challenges involved in component-based simulation. The component models may or may not have been developed with the federation in mind. The principal technical challenge that must be addressed "in dealing with this comprehensive and critical void include modeling and simulation composability, semantic interoperability and information sharing, and model composition at multiple levels of abstraction" [Benjamin, 2006].

An ontology-driven approach to component-based simulation includes the following steps: (a) component ontology building up; (b) a repository creation; (c) repository access to the M&S community providing. These steps suppose next activities:

(a) Define an ontology for all components that explicitly captures the definition of goals of the component, detailed descriptions of net inputs required for execution, detailed descriptions of total outputs generated, system requirements, constraints on successful execution, and preconditions and post conditions, etc.

(b) Create a virtual repository of components and component ontologies.

(c) An M&S expert attempting to compose a federation must be able to easily identify, based on ontology descriptions, the components required for the composition. The repository must allow for the downloading of component copies for such uses.

Almost all papers consider the process of creating ontologies and discuss the special languages such as OWL (a language for OWL ontologies building) [Dean, 2002], [Lacy, 2004]. The benefits of ontology management methods and tools, the role of ontology-based analysis and the architecture of OSFM are described in [Benjamin,2006] . OSFM is an Ontology-driven Simulation Modeling Framework (OSMF) solution that provides a "visual programming environment" to rapid compose, build, and maintain distributed, federated simulation. The role of ontologies in trajectory simulation is discussed. The ontology is regarded as the domain model component of the reuse infrastructure, and is being developed to be a reusable knowledge library on trajectory simulations [Durak, 2006].

The authors of paper [Liang, 2003] represent port ontology. It is suite program tool for simulation model composition with components (or subsystems) from repository. Port is an interface; this interface describes the boundaries of component (subsystem). System is a configuration of subsystems that are connected to each other through well-defined interfaces. The configuration interface of a component object consists of ports. Ports define the intended interaction between a component and its environment; interaction consists of the exchange of energy, matter or signals (information). For instance, the configuration interface of the motor may has ports for the stator, the shaft of the rotor, and the electrical connectors. It is through its ports that a component (sub-system) interacts with other components (sub-systems), as is indicated in the graph by the connection between ports. The fact that these interactions have been abstracted into ports does not imply that only components with standardized connectors can be defined in this fashion. When interfaces are not completely standardized (e.g. a weld between two structural elements), the interaction can still be abstracted into one of a relatively small set of general interactions types (e.g. a rigid mechanical connection).

Port ontology usage allows automating the process of model composition with appropriate components and subsystems because of the fact that ontologies save knowledge about ports connection.

Later we shall consider the representation of model in simulation system Triad, main features of this system, after that we will regard the example of partly described model and show the application of ontology approach to solve the problem of simulation model automatic completion.

Simulation system Triad.Net architecture

Let us consider simulation modeling system Triad.Net, its appointment, its components and functions of each component. So distributed simulation system Triad.Net is a modern version of previous simulation modeling system Triad [Mikov, 1995] dedicated to computer aided design and simulation of computer systems. Triad.Net is designed as distributed simulation system, so various objects of simulation model may be distributed on the different computer nodes of a computer system. One more specific characteristic of Triad.Net – remote access, so several investigators may fulfill a certain project from different computers situating in different geographical points.

Distributed simulation system Triad.Net consists of some subsystems: compiler (TriadCompile), core of simulation system (TriadCore), graphical and text editors, subsystem of testing and debugging (TriadDebugger), subsystem of distributed simulation (synchronization of simulation model objects which are situated on different calculating nodes of computer system, conservative and optimistic algorithms realization), subsystem for equal workload of calculating nodes (TriadBalance), subsystem of remote and local access (TriadEditor), subsystem of automatic and semiautomatic simulation model completeness (TriadBuilder). Components of simulation system Triad.Net are represented on fig. 2. Triad.Net is suitable for computer aided design of computer system and may be applied in various domains.

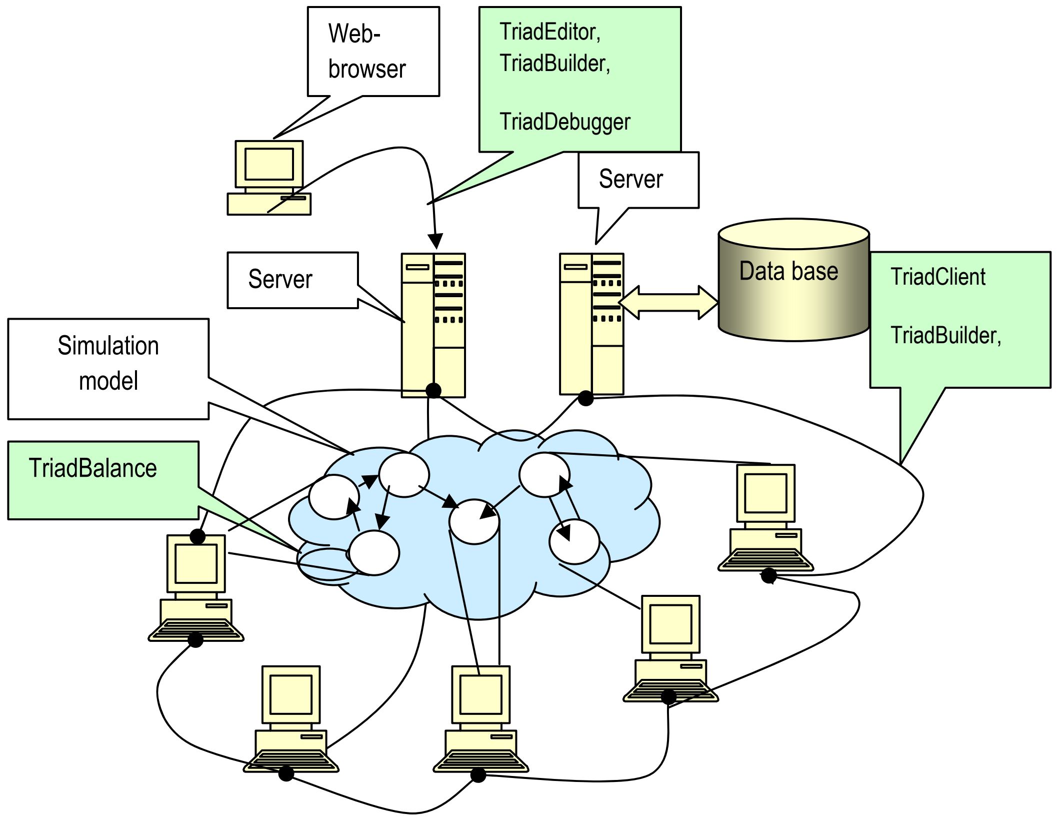


Fig.2. Simulation modeling system Triad.Net architecture

Now we are going to consider in details one of above components – the subsystem of automatic and semiautomatic simulation model completeness (TriadBuilder). Initially we address to the specific characteristics of simulation model in Triad.Net.

Model description

A simulation model μ = {*STR*, *ROUT*, *MES*} consists of three layers, where *STR* is a layer of structures, *ROUT* – a layer of routines and *MES* – a layer of messages.

The layer of structures is dedicated to describe the physical units and their interconnections, but the layer of routines presents its behavior. Each physical unit can send a signal (or message) to another unit. So each object has input and output poles. Input poles are used for sending messages. Output poles serve for receiving messages. A message of simple structure can be described in the layer of routines. A message of a complex structure can be described in the layer of messages only. Many objects being simulated have a hierarchical structure. Thus their description has a hierarchical structure too. One level of the system structure is presented by graph *P* = {*U*, *V*, *W*}. P-graph is named as graph with poles. *V* is a set of nodes, presenting the physical units of an object to be designed, *W* – a set of connections between them, *U* – a set of external poles of a graph. Internal poles of nodes are used for information exchange within the same structure level; in contrast the set of external poles serves to send signals (or more complex information) to the objects situated on higher or underlying levels of description.

Poles are very important part of a model. They represent “interfaces” of model components: objects, routines and graphs: communication links are being established through the poles of structure nodes; applying routine instance to a structure node consists of relating set of routine’s poles to the set of node’s poles; also, to complete operation of opening of structure node with a graph we need to relate outer poles of a graph to the poles of node (outer pole of a graph is a set of poles of it’s nodes used to communicate with the rest of the model). Thus, when a node is opened with a graph or routine is set for a node, this node or routine is “inserted” into model and interacts with the rest of the model through the “interface” described by poles of the node object.

A set of routines is named as routine layer *ROUT*. Special algorithms – elementary routines – define a behavior of a physical unit and are associated with particular node of graph *P* = {*U*, *V*, *W*}. Each routine is specified with the set of events (*E*-set), the linearly partly ordered set of time moments (*T*-set) and the set of states {*Q*-set}. Each state is specified with the values of local variables. Local variables are defined in each routine.

The state is changed only if any event occurs. One event schedules another event(s). Routine (as an object) has input and output poles too. An input pole serves to receive messages, output – to send them. A special statement *out* (*out* <message> *through* <pole name>) is used to send a message. An input event e_{in} has to be emphasized among the other events. All input messages are processed by the input event, and output messages – by the ordinary events.

System Triad.Net [Mikov, 2003] is advanced discrete-event simulation system Triad [Mikov, 1995], but it is the distributed/parallel one. Conservative and optimistic algorithms were designed in Triad.Net. Besides, Triad.Net is characterized by the following [Mikov, 1995]:

– Triad language includes the special type of variables – type “model”. There are several operations with the variable type “model”. The operations are defined for the model in general and as well as for each layer. For example, one may add or delete a node, add or delete an edge (arc), poles, create a union or an intersection of graphs. Besides, one or another routine (routine layer) using some rules can be assigned to the node (structure layer). The behavior of the object associated with this node would be changed. Besides, there's no need to retranslate the model. Thus a simulation model can be described by linguistic structures or built as a result of a model transformation algorithm.

– A simulation model is hierarchical, so each model (node) in a structure layer can be associated with some substructure.

– The model analysis subsystem has to provide a user with the possibility to formulate not regulated request. So the investigator may avoid the information superfluity or its insufficiency. The investigator can change the set of collected data within the simulation run, but model remains invariant. The model analysis subsystem has to posses smart software tools to analyze the simulation run results and to recommend the policy for the following simulation runs.

Partly described model

An ordinary simulation system is able to perform a simulation run for a completely described model only. At the initial stage of designing process an investigator may describe a model only partly omitting description of behavior of a model element $\mu_r^* = \{STR, ROUT^*, MES\}$). Simulation model may be described without any indication on the information flows effecting the model ($\mu_s^* = \{STR^*, ROUT^*, MES\}$) or without the rules of signal transformation in the layer of messages ($\mu_m^* = \{STR, ROUT^*, MES\}$). However for the simulation run and the following analysis of the model all these elements have to be described may be approximately.

For example, in a completely described model each terminal node $v_i \in V$ has an elementary routine $r_i \in ROUT$. An elementary routine is represented by a procedure. This procedure has to be called if one of poles of node v_i receives a message. But some of the terminal nodes v_i of partly described model do not have any routines. Therefore the task of an automatic completion of a simulation model consists either in "calculation" of appropriate elementary routines for these nodes, i.e. in defining $r_i = f(v_i)$, either in "calculation" of a structure graph $s_i = h(v_i)$ to open it with (in order to receive more detailed description of object being designed). It was mentioned above that the routine specifies behavioral function assigned to the node, but the structure graph specifies additional structure level of the model description. And at the same time, all structures s_i must be completely described as the sub-models.

These actions have to be fulfilled by the subsystem TriadBuilder. Fig.3 represents processing of simulation model in Triad.Net.

One may differentiate automatic and semiautomatic completeness of partly described simulation system. Semiautomatic completeness supposes to use a special linguistic construction and appropriated program component. It is "*conditions of simulation*". Another strategy: to put simulation model into the «environment of simulation».

Semiautomatic supposes that investigator have to include some statement in the part '*initial*' of '*conditions of simulation*'. We can denominate these statements: (a) an imposition of a routine on a node; (b) an imposition of a message layer on a model; (c) node description with substructure; (d) statements changing the structure of simulation model (adding and removing of some nodes, arcs, inputs and outputs and so on). One may use statement *simulate* to initiate simulation run. But at first simulator fulfill completion of partly defined simulation model (processing of statements in part *'initial'* of linguistic construction *'conditions of simulation'*). So, the semiautomatic completeness allows changing the behavior of the simulation model during the same simulation experiment (node description with substructure, an imposition of a routine on a node). Moreover semiautomatic completeness allows changing the structure of messages (an imposition of a message layer on a model). The information flow impact on model and algorithms of signal transformation may be changed if investigator will use another 'conditions of simulation'. Let us consider an example of simulate statement*: simulate M on condition of simulation New_Condition*(*M.N1.a*, *M.N2.b*), where *M.N1.a*, *M.N2.b* – are actual arguments. These actual arguments are the simulation model variables being under monitoring by information procedures of Triad.Net.

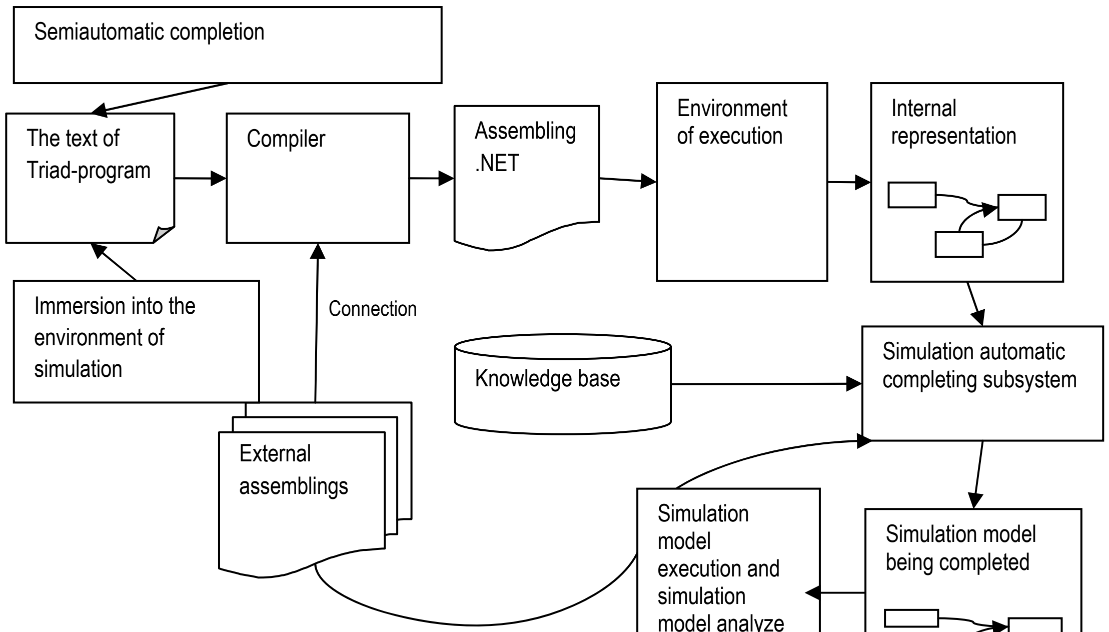

Semiautomatic completion
The text of Triad-program
Compiler
Assembling .NET
Environment of execution
Internal representation
Immersion into the environment of simulation
Connection
Knowledge base
Simulation automatic completing subsystem
External assemblings
Simulation model execution and simulation model analyze
Simulation model being completed

Fig 3. Simulation system Triad.Net and model completion subsystem

Let's consider an example of computer network fragment simulation. This fragment consists of workstations and routers (fig.4.). Each workstation has one neighbor (router). Workstations attempt to transfers data to another one. Data have to pass some routers, but the behavior of routers is unknown. So it is necessary to detect all nodes of simulation model, find out nodes (v_i) without routines, search out the appropriate one in data base of routines and fulfill the completion of model. Let's discuss the method of model completion suggested by authors.

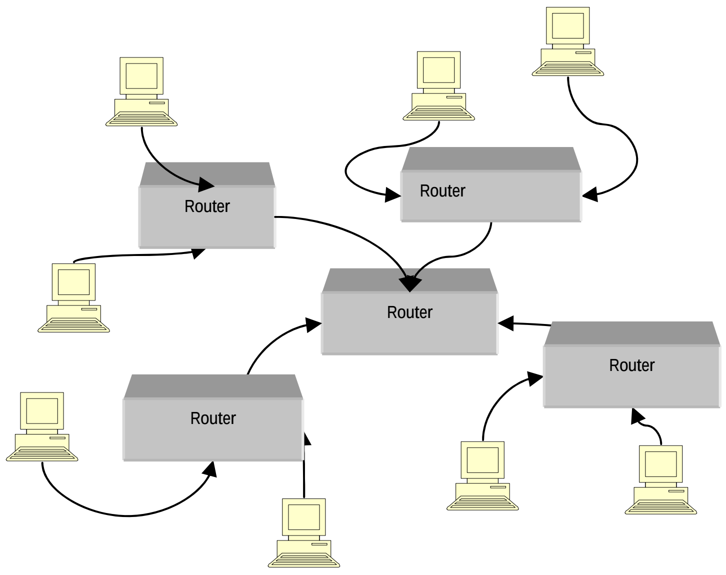


Fig. 4. The fragment of computer network consists of workstations and routers

The method of model completion

The method of a model *M* completion at a node *v* is based on following.

Node *v* is represented by sets *In*(*v*) and *Out*(*v*) of input and output poles respectively, its name and semantic type. Its poles are described by types of messages passing through them. Node *v* is located in some "environment" of the model *M*, it is adjacent to certain nodes v_k of the model. Nodes v_k for their part are also described by their poles, names, semantic types and environments.

All this information imposes restrictions on possible routine for *v*. It must not be chosen randomly. Despite some remaining freedom of choice, it should be picked out from some limited set.

An elementary routines library is created for a given simulation domain. The library should be consistent with domain ontology.

The "semantic type" concept is used to provide means for selection of an appropriate objects "opening". A semantic type of the node defines a possible object class from certain domain. The given node can represent objects of this class in a model. For example, when computing systems are simulated, one can define such semantic types as "processor", "register", "memory unit". Or, when queuing systems are simulated, such semantic types as "queue", "server" and "generator" would be appropriate. Using information, obtained from semantic type of a node, one can pick out an appropriate specification for it.

It is necessary to use special statement <object name> => <semantic type name> in order to declare semantic type of an object.

Semantic type could be declared by statement type <semantic type name>.

The fragment of Triad program can be given below. This fragment illustrates the statements mentioned above (to declare semantic type and to denote semantic type).

```
Type Router,Host;
integer i;
  M:=dStar(Rout[5]<Pol[4]>);
  M:=M+node Hst[8]<Pol>;
  M.Rout[0]=>Router;
  for i:=1 by 1 to 4 do
          M.Rout[i]=>Router;
          M:=M+edge(Rout[i].Pol[1]—Hst[2*i-2]);
          M:=M+edge(Rout[i].Pol[2]—Hst[2*i-1]);
  endf;
  for i:=0 by 1 to 7 do
          M.Hst[i]=>Host;
  endf;
M:=M+edge(Rout[1].Pol[2]--Rout[4].Pol[0])+edge(Rout[0].Pol[2]--Rout[3].Pol[0]);
```

Fig. 5. The fragment of Triad program (Program 1) with the semantic types.

An internal form of the simulation model can be gained as a result of a program run (fig.3.): it is a graph, each node of the graph represents a workstation or a router. Each workstation has a semantic type “host”, each router – semantic type “router”.

Corresponding routine instance search

So called specification, configuration and decomposition conditions are used to test the routine instance for each undefined node in order to complete simulation model. First of all, consider a specification condition. Closely related to specification condition is such a concept as “semantic type”. This concept was discussed earlier.

Let us assume that *v* – terminal node, *r* –routine instance from the knowledge base. Let us introduce the function *equtype*(*v, r*), defining specification condition performance. The result of this function is true, if the semantic type *Type*(*v*), assigned to the node corresponds to a semantic type *Type*(*r*), associated to routine instance found in the knowledge base.

Semantic type *T*1 corresponds to semantic type *T*2, if *T*1 is a superclass *T*2 (i.e. $T2 \subset T1$).

By this means the condition of specification is true if found routine instance corresponds to the semantic type of the node or to more special type.

Some specific type Object is introduced to provide the processing of the objects with unknown semantic type. Semantic type Object is a parent to all other semantic types. Any node is considered to be belonging to this parent type Object. If several particular semantic types are denoted the node is marked as belonging to each of them. Therefore routine instance without specified semantic type (more precisely, semantic type Object is specified) can be applied only to the node without definite semantic type. However, only routine instances belonging to the intersection of several semantic types can be applied to the nodes with these several types. For example, if we want to describe the node with a several functions (working as a router and host at the same time) we should declare both semantic types for it. In this case routine instances belonging only to one of these types would not be sufficient. We should find routine instance belonging to the intersection of these types, i.e. implementing both router and host functions.

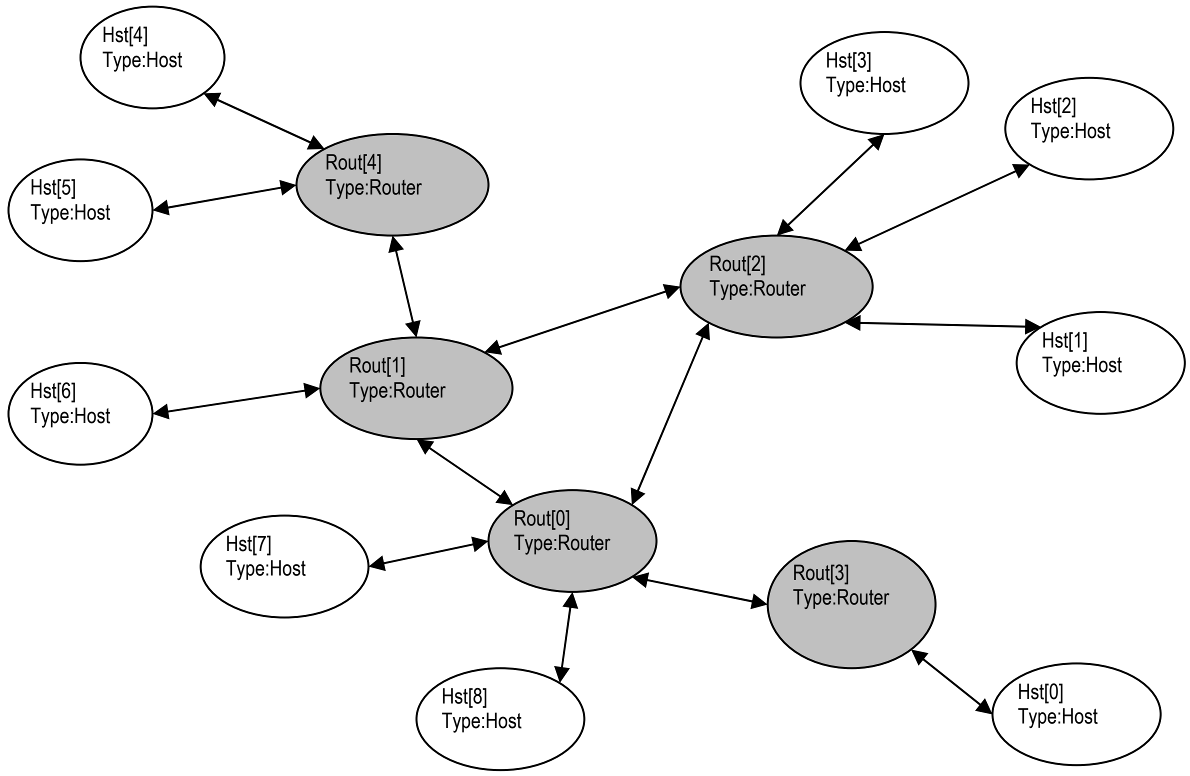


Fig.6. The internal form of computer network and semantic types

Configuration condition supposes the following: the amount of input and output poles of a node has to be equal to the amount of input and output poles of an appropriate routine instance. When we apply routine *r* to the node v the following relations are defined:

$$L_i: In(v) \rightarrow In(r) \tag{1}$$

$$L_o : Out(v) \rightarrow Out(r) \tag{2}$$

It is significant that these mappings are not functional ones. They define a set of related pairs (p_1, p_2), where $p_1 \in v$, $p_2 \in Pol(r)$, thus each pole can belong to any number of pairs or not to be used at all. Let $D(L_i/o)$ be cardinal numbers of input and output poles of a node, related to mappings *Li* and *Lo* respectively. Depending on these values the undefined node has to have some definite amount of input and output poles, to be more precise, the following mappings are significant:

$$|In(v)| \geq |D(L_i)| \tag{3}$$

$$|Out(v)| \geq |D(L_o)| \tag{4}$$

Nonrigourous equation allows using the nodes with an excessive amount of input and output poles when the routine instance is applied. The presence of the necessary minimum of poles is tested only, so some of inputs and outputs can be left “hanging”, and all the messages, sent through them will not be processed.

Nodes *Rout*[1 to 4] are declared as nodes with 4 poles in our example (*M* := *dStar*(*Rout*[5] < *Pol*[4]>);). However, each of them is connected only with 3 neighbors, so one of its poles will not be used.

Decomposition condition defines rules of node connections in a model graph, and is derived from node adjacency relations.

To define decomposition conditions we can introduce the concept of surrounding graph of some node *v*. Let $G = \{V;W;U\}$ be graph and node ($v \in V$) belonging to graph *G*. The relation *S* determines the adjacency of nodes in graph *G*, i.e.:

$$\forall v_1, v_2 \in V : (v_1; v_2) \in S \leftrightarrow \exists p_1 \in v_1, \exists p_2 \in v_2 : ((p_1; p_2) \in W \vee (p_2; p_1) \in W) .$$

Function Sub(w,v) defines a set of poles of the node w, connected with poles v:

$$Sub(w, v) = \{p \in w \mid \exists p_0 \in v : (p; p_0) \in W \vee (p_0; p) \in W\} \tag{5}$$

Then graph $GG(v) = \{V';W';\varnothing\}$ - is a surrounding graph for v if the following conditions are observed:

$$V' = \{v\} \cup \{w' \mid w' = Sub(w,v);\ (w;v) \in S\} \tag{6}$$

$$\forall w : (w;v) \in S \rightarrow Sub(w,v) \in V' \tag{7}$$

$$\forall p_1, p_2 : [(p_1;p_2) \in W] \wedge [p_1 \in v \vee p_2 \in v] \rightarrow (p_1;p_2) \in W' \tag{8}$$

$$W' \subset W \tag{9}$$

Eq (6) limits the set of nodes of the surrounding graph of the node *v*: it includes the node *v* itself and subsets of nodes adjacent with it. It is significant to take into account only those poles of adjacent nodes which are in accordance with Eq (5) directly connected with poles of node *v*.

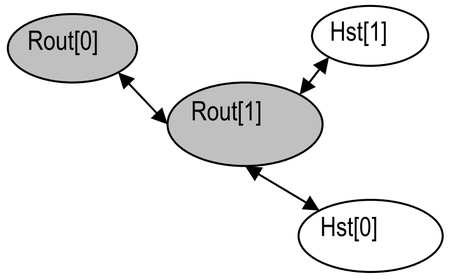


Fig.7. The surrounding graph of a node Rout[1]

Eq (7) specifies that this sort of subset exists for each adjacent node.

Eq (8) informs that each edge adjacent to the node v would be included to the surrounding graph. Eq (6) and Eq (7) state that all adequate poles would be included in the surrounding graph too.

Eq (9) asserts that there are no any excessive edges in the surrounding graph but only edges corresponding to Eq (8) are included.

So decomposition condition checks up following:

– which semantic types are set to the nodes connected to given node *v*,
– an isomorphism of two graphs: the surrounding graph of a node *GG*(*v*), and the pattern graph *GG*'(*r*) taken from the domain ontology.

The pattern graph is stored in a knowledge base and associated with a routine instance.

The fulfillment of decomposition condition is determined by function *iso*(*GG*'(*r*), *GG*(*v*)), which searches the environment graph of node *v* for a subgraph isomorphic to *GG*'(*r*), and also checks some additional restrictions for environment graph.

Graph *GG*'(*r*) is a pattern graph taken from the knowledge base and it should be relevant to the actual surrounding graph. If *v*' is a central node of this pattern graph then it shouldn't have any "hanging" poles, i.e. each pole of node *v*' corresponds at least one pole of routine.

Thus the task of model completion subsystem implies the following: to find the proper routine instance from the knowledge base for each undefined node in a partly described model. Therewith the conditions of specification, configuration and decomposition have to be followed. The information required to test these conditions should be stored in knowledge base. The ontology approach is used to represent this information.

Base ontology

The base ontology describes a model representation in Triad.Net: classes such as Model, Object, Routine, Polus and so on are specified in it.

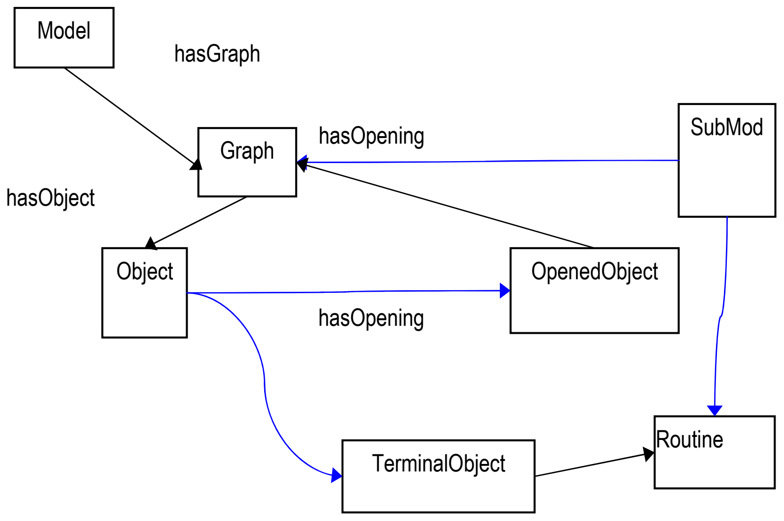


Fig.8. The fragment of ontology

Special class SubMod is presented describing everything the model object can be specified with. Its subclasses are the Routine and Graph classes representing the set of routines and structure graphs of one hierarchical level. Their common superclass allows unifying operations of opening object with a graph and applying the routine to an object.

The Web Ontology Language (OWL) is used to store the base ontology and all the domain ontologies. It was chosen because it allows composing information from different sources and is widespread and well known.

A part of the base ontology including most common concepts of ontology is depicted on Fig.5. “Has_subclass” relations are shown as blue arrows.

The poles hierarchy and its connection to the rest of ontology are missing on this picture.

Each semantic type defines a structure graph or a routine suitable for opening.

In the view of ontology knowledge representation semantic type is some class of nodes possessing certain properties. For example “request generator” is a node having at least one output pole connected to a node with semantic type of “queue”. Possible subclasses of “request generator” are generators of specific requests. To define them one can apply restrictions on message types corresponding to generator’s output only.

Thus, for any specific domain we should create ontology expanding the basic one which should contain information on how one should model the concepts from this domain.

Since we are mostly using the *Has_subcalss* relation it is convenient to have a hierarchy representation of an ontology. With the use of this approach, child nodes will represent concepts specifying concepts of parent nodes.

One could bind the most common concepts of a domain, for example, “device” when modeling the computing systems domain to root nodes. The nodes residing on lower levels of hierarchy would represent classes defining more precise concepts of a domain: devices would divide to “processor”, “memory unit” etc. “Processors” would be divided depending on their architecture and so on.

Simulator subsystem for model completion

The model completion subsystem includes the following components:

– Model analyzer. It searches through model and looks for terminal nodes without routines to mark them as needed to process.
– Model converter. It converts model from its internal form to the ontology representation, it is used to save surrounding graph of a routine instance.
– Inference module. It analyses the model ontology and searches it for an appropriate routine instances for each node marked by model analyzer.
– Model builder. Applies found routine instances to the nodes.
– Knowledge base, containing information about semantic types and routine instances with their pattern graphs.

Let us consider the algorithm of model completion subsystem in our example (fig. 3).

Model completion subsystem starts when the internal form of simulation model is built according to a Triad code.

First, model analyzer searches the model for incomplete nodes, and marks them. Assume that only the router nodes don’t have routines applied. Thus, the model analyzer will mark all *Rout* nodes.

The inference module starts looking for an appropriate routine instance for each of marked nodes. We’ll take *Rout*[1] as an example. Assume, that there are several routine instances for a Router semantic type, describing routers with 2, 3… 10 neighbors.

According to specification condition, inference module picks out these 9 routine instances discarding the ones with other semantic types.

Then, according to configuration condition it discards the instances of a router routine with 5 or more neighbors (*Rout*[1] is declared having 4 poles).

Lastly, it starts checking decomposition condition for remaining routine instances. Instance with 4 neighbors will be discarded, because the surrounding graph of *Rout*[1] has only 4 nodes, and the pattern graph for the instance has 5 nodes. All others will suffice for the condition. In order to avoid ambiguity and to choose the most appropriate instances, they are sorted before checking the decomposition condition, according to several heuristics (by the number of nodes for example).

After the appropriate instance has been found, it is applied to the node.

Conclusions

The paper considers a problem of completeness of partly defined simulation model. Partly defined simulation model and it's analyze and investigation may be actual at the beginning of simulation process. An investigator may not know all details of simulation model being under design. For example, an investigator cannot describe in detail the behavior of some components of computer network. But he wants to receive some results while simulations run. So it is necessary to complete simulation model description. Authors suggest automatic and semiautomatic completeness. Automatic and semiautomatic completeness of partly defined simulation model are the function of special component (TriadBuilder) of simulation system Triad.Net.

Semiautomatic completeness supposes special linguistic construction ("conditions of simulation") and appropriate program component using. One can change the initial part of this construction, more precisely, to change some statements in this part. These statements may change the behavior of some simulation model components or it's structure.

The automatic completeness allows searching program components in special data bases following some criteria and thanks to knowledge derived from ontologies.

Ontologies are the convenient path to domain describing. They are efficient for information systems design, data bases and complex programming systems development, computer network design and so on. It is a powerful tool for the system designers. Ontology can be useful to system researcher too. Investigation of a complex system by simulation methods together with ontology allows him/her to get results in difficult cases of incomplete, fuzzy models.

Bibliography

[Mikov,1995] Simulation and Design of Hardware and Software with Triad// Proc.2nd Intl.Conf. on Electronic Hardware Description Languages, Las Vegas, USA, 1995. pp. 15-20.

[Mikov,1995] Mikov A.I.. Formal Method for Design of Dynamic Objects and Its Implementation in CAD Systems // Gero J.S. and F.Sudweeks F.(eds), Advances in Formal Design Methods for CAD, Preprints of the IFIP WG 5.2 Workshop on Formal Design Methods for Computer-Aided Design, Mexico, 1995, pp. 105 -127.

[Mikov,2003] Mikov A.I., Zamyatina E.B., Fatykhov A.H.. A System for Performing Operations on Distributed Simulation Models of Telecommunication Nets // Proc. I Conf. "Methods and Means of Information Processing, Moscow State University (Russia), 2003. pp. 437-443 (in Russian).

[Fishwick,2004] Fishwick P.A. Ontologies For Modeling And Simulation: Issues And Approaches /Paul A. Fishwick, John A. Miller // Proceedings of the 2004 Winter Simulation Conference. pp. 259-264

[Dean,2002] Dean M., Connolly D., van Harmelen F., et al. 2002. Web Ontology Language (OWL) Reference Version 1.0. W3C. //www.w3.org/TR/2002/owl-ref/

[Silver,2006] Silver G.A, Lacy. L.W., J.A. Miller. Ontology Based Representations Of Simulation Models Following The Process Interaction World View. Proceedings of the 2006 Winter Simulation Conference L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds., pp.1168-1176.

[Durak,2006] Durak U., Oguztuzun H., Ider S. K..An Ontology For Trajectory Simulation. Proceedings of the 2006 Winter Simulation Conference/ L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds. pp.1161-1167

[Benjamin,1995] Benjamin P., Menzel C, Mayer R. J. Towards a method for acquiring CIM ontologies. // International Journal of Computer Integrated Manufacturing, 8 (3) 1995, pp. 225-234.

[Miller,2005] Miller J.A., Baramidze G. Simulation and the Semantic Web // Proceedings of the 2005 Winter Simulation Conference / M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.. – pp. 2371-2377

[Benjamin,2006]Benjamin P., Patki M., Mayer R. J. Using Ontologies For Simulation Modeling // Proceedings of the 2006 Winter Simulation Conference/ L. F. Perrone, F. P. Wieland, J. Liu, B. G. Lawson, D. M. Nicol, and R. M. Fujimoto, eds. –pp.1161-1167

[Benjamin, 2005] Benjamin P., Akella K.V., Malek K., Fernandes R. An Ontology-Driven Framework for Process-Oriented Applications // Proceedings of the 2005 Winter Simulation Conference / M. E. Kuhl, N. M. Steiger, F. B. Armstrong, and J. A. Joines, eds.,– pp 2355-2363

[Rathnam,2004]Rathnam T., Paredis C.J.J. Developung Federation Object Models Using Ontologies // Proceedings of the 2004 Winter Simulation Conference / R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.,– pp 1054-1062

[Gustavson,2004] Gustavson P., Chase T. Using XML and BOMs to Rapidly Compose Simulations and Simulation Environments // Proceedings of the 2004 Winter Simulation Conference / R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.,– pp. 1467-1475

[Lacy,2004] Lacy L.. Potential Modeling And Simulation Applications Of The Web Ontology Language – Owl / Lee Lacy, William Gerber // Proceedings of the 2004 Winter Simulation Conference. pp. 265-270

[Liang,2003] Liang V-C. A Port Ontology For Automated Model Composition / Vei-Chung Liang, Christiaan J.J. Paredis // Proceedings of the 2003 Winter Simulation Conference, pp. 613-622

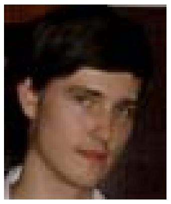
Authors' Information

Alexander Mikov – Cuban State University, Head of Department of Computer Technologies, Aksaiskaya str., 40/1-28; e-mail: alexander_mikov@mail.ru

Major Fields of Scientific Research: General theoretical information research, Information Systems, Simulation, Distributed and Parallel programming, Software technologies

Elena Zamyatina – Perm State University, Department of Software of Computer Systems, 614017, Turgeneva str., 33–40, Perm, Russia; e-mail: e_zamyatina@mail.ru

Major Fields of Scientific Research: Simulation, Distributed and Parallel programming, Multiagent Systems



Evgenii Kubrak –Perm State University, Postgraduate of Department of Software of Computer Systems, Leonova str,.20-5, Perm, Russian Federation; e-mail:q_brick@mail.ru

Major Fields of Scientific Research: Simulation, Software technologies, Information systems, Multiagent systems