IMPLEMENTATION OF SIMULATION PROCESS UNDER INCOMPLETE KNOWLEDGE USING DOMAIN ONTOLOGY

Alexander Mikov 1 , Elena Zamyatina 2 , Evgeniy Kubrak 3

1 Computing Research Institute,
614000 Perm, Sibirskaya 30-58, Russia
2 Perm State University, Faculty of Mathematic and Mechanic
614000 Perm, Bukirev 15, Russia
3 Perm State University, Faculty of Mathematic and Mechanic
614000 Perm, Bukirev 15, Russia

e_zamyatina@mail.ru (Elena Zamyatina)

Abstract

The problem of complex systems (information systems, computer networks, for example) analysis under uncertain conditions is discussed. This situation arises because of model incompleteness. Usually the behavior of some simulation model elements is unknown. 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. The paper is concluded by describing the programming tools which provide an ontology-based automatic completion of partly described simulation model in simulation system Triad.

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

Presenting Author's biography

Elena Zamyatina is an Associated Professor of Computer Science at Perm State University, the lecturer of Mathematical Department. Her research interests include simulation as well as parallel and distributed systems. She received a Ph.D. degree in Information and Computer Science in 1993. She has over 25 years of professional experience in programming tools design, particularly in simulation domain. She is the author of over 60 papers in the area of simulation, distributed simulation, parallel and distributed systems.

ISBN 978-3-901608-32-2

1 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 database search will take and will it be successful (information system is an object of investigations). 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 can not 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 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 problem solving used in simulation system Triad.Net [1,2,3]

2 Related works

Some papers, dedicated to motivations for using ontologies in simulation modeling and role of ontologies in simulation process, appeared last time [4,5,6,7]. Almost all papers consider the process for creating ontologies and discuss the special languages such as OWL (a language for OWL ontologies building). So the process interaction DES domain is discussed, and an approach to ontology based simulation model representation is presented in [8]. The benefits of ontology management methods and tools, the role of ontology-based analysis and the architecture of OSFM describe in [9]. OSFM is an Ontology-driven Simulation Modeling Framework (OSMF) solution that provides a "visual programming environment" to rapidly compose, build, and maintain distributed, federated simulation. The role of ontologies in trajectory simulation is discussed [10]. 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.

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

3 Model description

A simulation model $\mu=\{STR, ROUT, MES\}$ 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

ISBN 978-3-901608-32-2

in 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 [3] is advanced discrete-event simulation system Triad [1, 2], but it is the distributed/parallel one. Conservative and optimistic algorithms were designed in Triad.Net. Besides, Triad.Net is characterized by the following [1, 2]:

- 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.

4 Partly described model

An ordinary simulation system is able to perform a simulation run for a completely described model only.

As it was described above 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 have no 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 submodels.

Let's consider an example of computer network fragment simulation. This fragment consists of workstations and routers (fig.1.). 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.

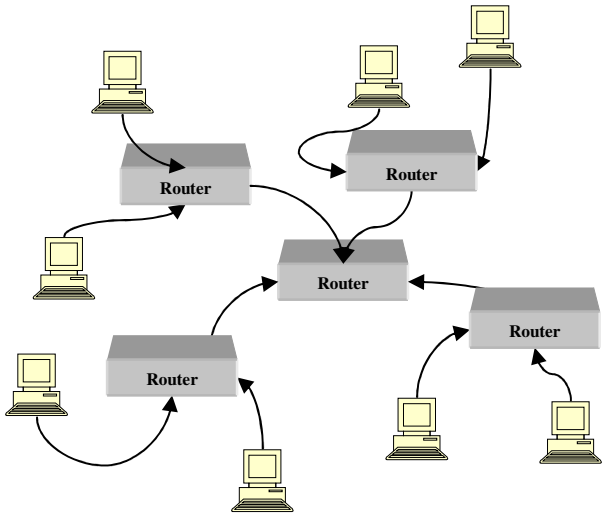

Fig. 1. The fragment of computer network consists of workstations and routers.

5 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.

ISBN 978-3-901608-32-2

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;
```

Fig. 2. 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".

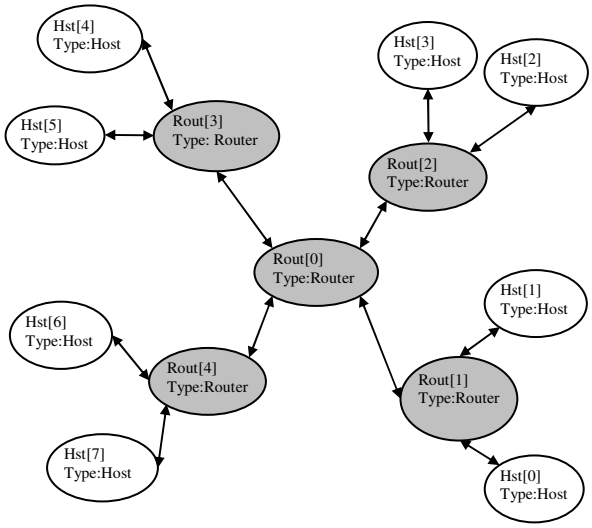

Fig.3. The internal form of computer network and semantic types

6 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 a 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. $T_2 \subset T_1$).

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.

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:

$$L_i : In(v) \to In(r) \quad (1)$$

$$L_o : Out(v) \to Out(r) \quad (2)$$

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 L_i and L_o 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:

$$\left|In(v)\right| \geq \left|D(L_i)\right| \quad (3)$$

$$\left|Out(v)\right| \geq \left|D(L_o)\right| \quad (4)$$

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*:

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

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

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

$$\forall w : (w; v) \in S \to Sub(w; v) \in V' \quad (7)$$

$$\forall p_1, p_2 : [(p_1; p_2) \in W] \wedge [p_1 \in v \vee p_2 \in v] \to (p_1; p_2) \in W' \quad (8)$$

$$W' \subset W \quad (9)$$

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*.

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.

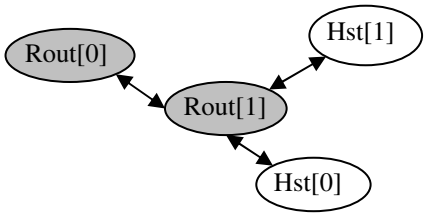

Fig.4. The surrounding graph of a node Rout[1]

So decomposition condition checks up following:

1) which semantic types are set to the nodes connected to given node *v*,

2) 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.

ISBN 978-3-901608-32-2

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.

7 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.

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.

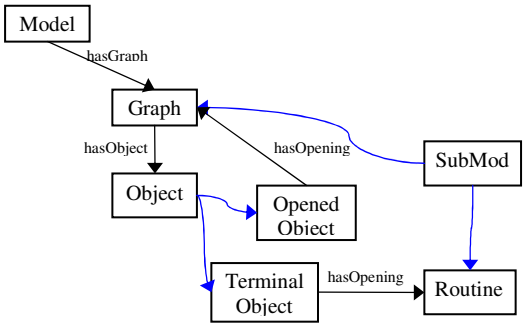

Fig.5. The fragment of ontology

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.

8 Simulator subsystem for model completion

The model completion subsystem includes the following components:

1. Model analyzer. It searches through model and looks for terminal nodes without routines to mark them as needed to process.

2. Model converter. It converts model from its internal form to the ontology representation, it is used to save surrounding graph of a routine instance.

3. Inference module. It analyses the model ontology and searches it for an appropriate routine instances for each node marked by model analyzer.

4. Model builder. Applies found routine instances to the nodes.

5. Knowledge base, containing information about semantic types and routine instances with their pattern graphs.

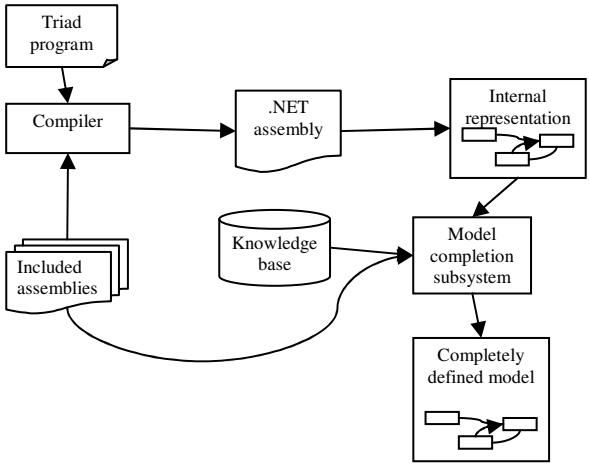

Fig 6. Simulation system Triad.Net and model completion subsystem

Let us consider the algorithm of model completion subsystem in our example (fig 1).

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.

9 Conclusions

Thus the suggested approach allows to automate the process of simulation model generation. This approach supposes use of 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.

Reference

[1] A.I. Mikov. 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.

[2] A.I. Mikov. 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.

[3] A.I. Mikov, E.B. Zamyatina, A.H. Fatykhov. 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).

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

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

[6] L. Lacy. 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

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

[8] G.A. Silver, L.W. Lacy, 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.

[9] P. Benjamin, M. Patki, R. Mayer. 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

[10] U. Durak, H. Oguztuzun, S. K. Ider. 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

