Proceedings of the 2004 Winter Simulation Conference
R .G. Ingalls, M. D. Rossetti, J. S. Smith, and B. A. Peters, eds.

KNOWLEDGE REPRESENTATION FOR CONCEPTUAL SIMULATION MODELING

Ming Zhou

Mechanical Engineering Technology
Indiana State University
Terre Haute, IN 47809, U.S.A.

Young Jun Son

Systems and Industrial Engineering
University of Arizona
Tucson, AZ 85721, U.S.A.

Zhimin Chen

College of Management
Shenzhen University
Shenzhen, Guangdong, 518060, P.R.C.

ABSTRACT

Simulation is a powerful tool that helps decision makers in business and industry to solve difficult and complex problems, reduce cost, improve quality and productivity, and shorten time-to-market. However the technology is still underutilized in many applications due to several reasons. In this study we address these issues using a knowledge engineering approach, i.e. develop efficient and robust models and formats to capture, represent and organize the knowledge for developing conceptual simulation models that can be generalized and interfaced with different applications and implementation tools. The research fits into a larger project effort that aims to create a sustained research program on knowledge-based simulation.

1 INTRODUCTION

Simulation has been recognized as one of the most powerful tools that help decision makers in manufacturing and other industries to solve difficult and complex problems for design, control or improvement of systems. The benefits from using simulation include reduced costs, improved quality and productivity, and shortened time-to-market. In spite of its power and benefits, the technology is still underutilized in many applications, and viewed, by many practitioners, as Alladin's lamp: -- a powerful but intimidating tool for use. The main reasons are: (1) simulation modeling is a time-consuming and knowledge intensive process requiring not only the knowledge from application domain, but also from the simulation and implementation domain (Arons 1999, 2000; McLean 2001). This cross-domain communication has caused great amount of difficulties in simulation modeling, and the cost for training and skill development is very high. (2) Most simulation models developed with the current technology are customized "rigid" models that cannot be reused or easily adapted to other even similar problems. (3) With the current technology, simulation modeling is still an *ad-hoc* process, i.e. a craft rather than a science. The modeling quality and efficiency depend largely on the skill and experience of human modelers (Mclean 2001). The loss of "intellectual capital" due to a high turnover rate and continuous retirement of experienced employees has further worsened the problem.

Limited efforts have been made in both academia and industries to address the difficulties that inhibit the deployment of simulation, primarily in the following aspects: (1) develop standard templates to address classes of simulation problems; (2) develop modularized models or component-based modeling approach; (3) develop standard interface that integrates simulation with other application systems; and (4) develop neutral data formats to facilitate model data transfer between different systems (Pidd 1998, Son 2003). Although there have been published results, the problems or issues are still far from satisfactory resolution.

Our study proposes a knowledge-based (KB) approach to address the difficulties of simulation modeling with the assistance of artificial intelligence (AI). We focus on the development of discrete-event simulation models (DES), and investigate robust and efficient representations of the modeling knowledge and artificial systems that process this knowledge to build valid models for DES applications. Robust representations capture the underlying logic and general functions of a simulation model and are independent of application and implementation specificities, and thus can be generalized to facilitate broader application of simulation. They can also be used to streamline and standardize the decision-making activities in simulation to improve the efficiency and reduce the barriers in modeling and analysis.

Knowledge-based simulation (KBS) has been considered as a promising approach to facilitate simulation model construction and execution. Early works of KBS can be described in three general categories: (1) developing "extended programming languages", i.e. general programming augmented with simulation oriented language constructs; (2) developing specialized simulation language based on a flow-chart type of logic; and (3) developing better interface to create a more interactive type (as opposed to "batch" type) of modeling environment. In the past two decades, the object-oriented representation of simulation concepts has been emphasized. For instance Fox (1989) used an object-oriented approach to developing schemata to capture and represent knowledge in appropriate forms to ease the creation of executable simulation models. Another approach focused on knowledge-based assistance in the implementation of simulation, e.g. developing executable model with specific commercially available tools (Arons 1999, 2000). In software vendor industry, emphasis has also been given to the development of high-level simulators for special application systems (McLean 2001; Zülch 2000). Given these developments, researchers and practitioners still face great challenges in KBS, particularly in the conceptual modeling phase. Very limited progress has been made to help modeling at conceptual level, and there is a significant gap between model conceptualization and implementation. We summarize the reasons as follows: (1) lack of knowledge-based assistance to translate or transform concepts from application domain to simulation domain during model conceptualization; (2) lack of in-depth study on the formulation and representation of conceptual simulation models (CSMs), e.g. robust representations that facilitate the specialization and generalization of modeling and flexible interface to a diversified implementation environment; (3) lack of rigorous analysis and definition of the relationship between the different formats of the representation to address the difficulties in man-machine communication; (4) lack of efficient mechanism to process and transform the knowledge and information to construct conceptual models and transform them into executable models.

In this paper we focus on the identification and representation of the knowledge for the conceptualization of DES models. In Section 2, we decompose a simulation modeling process to characterize the information transformation and knowledge utilization. In Section 3, we identify the basic set of concepts that can be generalized to many DES applications. Section 4 proposes a notation to formalize the knowledge identified. Rules and algorithms for developing conceptual model specifications are presented in Section 5. Section 6 gives an example for illustration, and we give conclusions and further research in Section 7.

2 MODELING: CONCEPTUALIZATION VS. IMPLEMENTATION

In general, we can categorize the knowledge used in a simulation modeling process into three domains: application, simulation, and implementation. The expressions for the same concept are usually different by domains. Consider expressions for a "product" concept that is an automotive engine block. In the application domain, it is labeled as "Engine block"; in the simulation domain: "Object" or "Product"; in the implementation domain, it is "Entity" or "Load" if we use Arena © and AutoMod © respectively. This cross-domain communication has caused great difficulty in simulation modeling. Most existing simulation software use representations or modeling approaches that mix the concepts of different domains and embed implementation requirements in conceptual modeling phase. This creates great difficulty for many practitioners who are "domain experts" but know little or nothing about simulation and implementation tools. From an information transformation point of view, a simulation modeling process can be decomposed into two sub-processes that transform data and information from one domain to another (see Figure 1). The first sub-process corresponds to model conceptualization. translating an application problem definition (APD) from the application domain into a simulation problem definition (SPD) in the simulation context. The SPD specifies the logical and structural requirements of a conceptual simulation model (CSM). This process uses a synthesized or generalized approach to map special concepts from a specific application into the general concepts of simulation, and results in a synthesized logical model independent of implementation requirements (e.g. not constrained by any specific simulation language). The second sub-process, "implementation", translates the logical and structural specifications of a CSM into a special or language-specific model. This process uses an entailed and implementation specific approach to produce an executable model with a specific software tool (called - "implemented simulation model"-(ISM)).

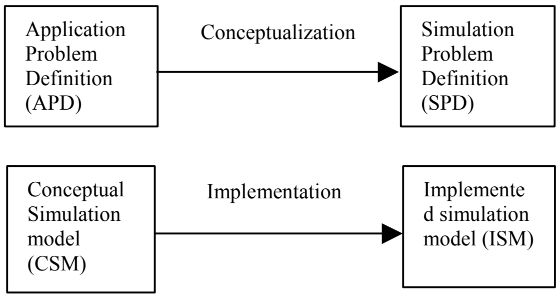
Application Problem Definition (APD)
Conceptualization
Simulation Problem Definition (SPD)
Conceptual Simulation model (CSM)
Implementation
Implemente d simulation model (ISM)

Figure 1: Conceptualization v.s. Implementation

We separate the knowledge used in conceptualization from that used in implementation so that modelers will not be constrained by implementation specifics during the conceptual modeling stage. This frees their ability to concentrate on the development of efficient, robust or generic representations that can be implemented by a variety of tools. From a knowledge utilization point of view, the two processes use different types of knowledge and reasoning styles. Here we consider that the knowledge includes theory (concepts and principles), skills (know-how), and experiences for simulation modeling. The conceptualization process uses more generalized and fundamental knowledge, while the implementation requires more specific knowledge related to the software tools selected. Essentially the conceptualization process redefines an application problem in the context of simulation requirement with an emphasis on the "system" (from which the problem was developed) and its behavior related to the problem. The reasoning strategies used in this generalized mapping include classification and abstraction. The classification performs a "pattern recognition" function that classifies input concepts into predefined categories. The abstraction, on the other hand, represents the concepts at appropriate levels to satisfy the efficiency of modeling. In general we can classify the knowledge used for conceptual modeling into two types: *application knowledge* and *simulation knowledge*. Application knowledge is a set of **special concepts** associated with a specific application domain, such as manufacturing, logistics and distribution or health-care systems. Special concepts are used to define and describe the unique characteristics of different application systems. They provide special knowledge and information required to build models for a specific type of application. Special concepts are used selectively in simulation models (see Figure 2), e.g. "conveyor" is a concept typically used by manufacturing type of models. Simulation knowledge, on the other hand, is a set of **basic concepts** that are shared by all DES models, and belong to the domain of simulation. To build a simulation model, we use concepts from both domains. In Figure 2, arrows represent the selection of concepts from a knowledge base to construct a certain type of models. A solid arrow indicates a required selection while a dashed arrow implies an optional selection depending on the application.

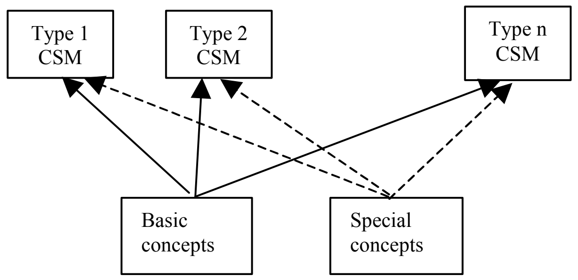

Figure 2: Two Sets of Modeling Concepts

Separating modeling knowledge in this way help us establish an effective strategy and focus on knowledge acquisition and robust representation, and facilitate the organization of the knowledge.

3 BASIC KNOWLEDGE FOR CONCEPTUAL SIMULATION MODELS

In this section, we focus on the identification of a common set of concepts that are shared by many discrete-event simulation models, in another word, these concepts can be generalized to many special applications of simulation. In doing so we followed a "process-oriented" approach to identify the related concepts. Different views of simulation often result in different modeling concepts. Comparing with other approaches (e.g. event-driven approach), a process-oriented view is easier for understanding and more descriptive, which is very important for knowledge based modeling, because a logic that is more "compatible" with human cognitive processes usually simplifies knowledge representation and inference. In addition, being more descriptive generally improves the modeling efficiency of simulation. According to Law (1991): "A process is a time-ordered sequence of interrelated events separated by passage of time, which describes the entire experience of an entity as it flows through a system". From a simulation perspective, a conceptual model consists of a set of concepts, defined at an abstraction level, that completely specifies the structure and behavior of a simulation model. The abstraction determines how much details should be visible to a modeler. Too much abstraction makes a model less descriptive or even useless, while too much detail increases modeling difficulty without improving model effectiveness. Unlike implementation modeling, whose abstraction level is dependent on the capability of modeling constructs provided by the tools (e.g. ARENA, AutoMod), conceptual modeling defines abstraction through proper representation schemes. Most practitioners use commercially available tools that use high-level modularized graphical commands, such as BLOCKS and MODULES. Therefore we need to identify modeling concepts that are "compatible" with the developing trend of software tools to reduce difficulties in translating from conceptual models to implementation models. In this study we propose modeling concepts that are classified in the following categories.

Objects: those that flow through or "fixed" within the system being modeled, such as parts, assemblies, orders, machine tools, etc. It can be divided into two types:

- **Entities**: those objects that flow through the system to receive services provided by a sequence of activities, e.g. parts and assemblies. Some software has used "external entity and internal entity" for implementation, we prefer not to make this differentiation. Functionally speaking, given an appropriate definition of entity and control logic, the needs for "internal entity" can be eliminated.
- **Resources**: those objects that are placed at fixed locations of a system according to certain configuration to provide means of service, e.g. operators, machine tools, inspection equipment, queues; and also mobile resources such as transporters.

Logical or functional activities: an "activity" is a logical process that performs a defined function required by simulation. These activities process entities, control or manipulate the flow of entities, or collect the data generated through the flow. Collectively this set of activities decompose the overall (functional) behavior of a simulation model, i.e. they partition the set of functions performed by the model. System states often change when entities flow through these activities. The activities are usually defined at abstract levels, and each is composed of several sub-activities, e.g. a processing activity may contain sub-activities such as seizing a resource, delaying for a time interval, and releasing the resource. The composite nature allows us to define simulation activities at an abstract level to encapsulate sub-activities to simplify modeling and represent the activities in a compact and organized hierarchy. Following is a classification of "top-level" activities based on their functions:

- Create entities;
- Change/assign entity properties/attributes;
- Process/service entities (processing activities, including inspection);
- Store/hold entities;
- Aggregate/disaggregate entities (e.g. assemble, disassemble or simply batch parts);
- Transport entities (move from one location to another);
- Branch the flow of entities (condition-based versus probability-based)
- Control the movement (temporal) of entities;
- Dispose entities;
- Collect data/statistics (counts, tallies)

A process flow (or control) logic: it defines the order of activities (from an entity-flow perspective) and the interactions between the entities and activities, and between the activities, for instance, order of processing activities as specified in a process plan; waiting for a control signal between processing activities. This logic specifies the flow and control of simulated objects in the model completely, but implicitly defines the events and state transitions that impact the behavior of the model. The logic serves as a basis for configuring the logical components to form a model. In general there are three types of the control of entity flow:

- Control the direction of the movement, i.e. routing entities. It does not stop the entity flow in simulated time. Three main approaches are:
    - **Sequential**: specified by a predetermined sequence defined in a process plan
    - **Condition**: determined by testing a predetermined condition
    - **Random sampling**: determined by the result from a random sampling (probabilistic routing).
- Control the timing of the movement, e.g. holding entities until a certain event occurs. It may stop entity flow in simulated time. Two main approaches are:
    - **Control by condition**: hold entities until a prescribed condition becomes true;
    - **Control by stimulus** (or control signal): hold entities until a predetermined type of signal is received.
- Control the quantity of entity flow, e.g. allowing a limited number of entities per release (of movement).

A process flow logic can be captured and represented in various forms, e.g. a "process narrative" in a textual form or a flow-chart diagram in graphical form. For instance, Kienbaum (1994) proposed to use a hierarchical activity cycle diagram (H-ACD) to represent the logic of discrete event simulation for object oriented design and development.

Data/input requirements: there are two types: (a) Numerical attributes or properties associated with objects and activities e.g. input distributions of entity arrival, processing time and transportation distance or time. (b) Global variables and expressions used to define and implement the process flow logic; including those called "control elements" by other researchers (Banks, 1998).

Output requirements: including the goals and objectives of simulation analysis, and the definitions on the expected results from simulation output analysis:

- Goals and objectives (e.g. comparing alternatives; estimating/predicting performance measures; analyzing sensitivity of input variables; optimizing system performance)
- Performance measures
- Confidence levels

Design of experiments: strategies and procedures to achieve what has been specified by "Output requirements", i.e. how to collect and analyze the data through simulation experiments? For instance:

- Type of analysis
- Plan or procedure of the analysis

These concepts provide knowledge needed for building conceptual simulation models and should be appropriately represented and formatted in a knowledge base (KB). Ideally they should be represented through a standard vocabulary and taxonomic structure. In addition to being a basis for requirement specification, these concepts also define the correspondence for mapping the knowledge from the application domain to the simulation domain.

4 SPECIFICATIONS OF CONCEPTUAL SIMULATION MODEL

In general the requirement specification of a conceptual model includes the basic concepts identified earlier and special concepts related to the application. These specifications should be formalized to support model development and validation, and facilitate the analysis of knowledge representations and inference procedures. In this study we propose the following sets of notation to formalize a CSM specification:

$\boldsymbol{CSM} = \langle \boldsymbol{S_1}, \ldots \ldots, \boldsymbol{S_m} \rangle$, i.e. CSM is a partition of sets $\boldsymbol{S_1}, \ldots \ldots, \boldsymbol{S_m}$, where each $\boldsymbol{S_i}$, $1 \le i \le m$, is a set of partial specification of CSM. The definitions are given as follows:

- $\boldsymbol{S_1}$ = Set of simulation objects; $\exists\, \boldsymbol{A_1} \leftrightarrow \boldsymbol{S_1}$, $\boldsymbol{A_1}$ = A set of attributes associated with $\boldsymbol{S_1}$. Symbol "$\leftrightarrow$" is used as an "association" operator here. $\boldsymbol{S_1} = \langle \boldsymbol{E}, \boldsymbol{R} \rangle$, where $\boldsymbol{E}$ = Set of entities and $\boldsymbol{R}$ = Set of resources; and $\exists\, \boldsymbol{A_E} \leftrightarrow \boldsymbol{E}$, and $\exists\, \boldsymbol{A_R} \leftrightarrow \boldsymbol{R}$, where $\boldsymbol{A_E}$ and $\boldsymbol{A_R}$ are sets of attributes. For instance, $\boldsymbol{A_E} = \{A_{E1}, \ldots, A_{En}\}$; where A_{E1} = Entity Name; A_{E2} = Entity Type; A_{E3} = Arrival Pattern = Inter-arrival time distribution; A_{E4} = Process Plan (Routing) = An entity-dependent sequence of processing or inspection operations $(O_1, \ldots, O_k)$; Set $\boldsymbol{E}$ or $\boldsymbol{R}$ can be further decomposed into subsets, e.g. $\boldsymbol{R} = \langle \boldsymbol{R_L}, \boldsymbol{R_E}, \boldsymbol{R_T}, \boldsymbol{R_Q} \rangle$; where: $\boldsymbol{R_L}$ = Set of labor requirements; $\boldsymbol{R_E}$ = Set of equipment requirements (including machines, tools, etc.); $\boldsymbol{R_T}$ = Set of transportation resources; $\boldsymbol{R_Q}$ = Set of waiting/staging spaces (queues).
- $\boldsymbol{S_2}$ = Set of logical activities = $\langle \boldsymbol{P_1}, \ldots, \boldsymbol{P_n} \rangle$ and $\exists\, \boldsymbol{A_2} \leftrightarrow \boldsymbol{S_2}$. These activities are identified and defined based on a process-oriented view: what entities see while they flow through the system from their "birth" to their "death" during the simulation. Note that this is the set of activities used by ALL types of entity, i.e. a specific entity type may not use all of them. $\boldsymbol{A_2} = \{A_{21}, A_{22}, \ldots, A_{2k}\}$, where the set of common attributes are defined as: A_{21} = Activity.Name; A_{22} = Activity.ID, i.e. a code used by model to identify the type of the activity; A_{23} = Activity.Resource; A_{24} = Activity.Delay; A_{25} = Activity.Control, i.e. a code that identify the type of control associated with this activity (e.g. sequential, assigning, branching, holding, combining, splitting, assembling, etc.) $\forall j$, $1 \le j \le n$, $\boldsymbol{P_j}$ is decomposable: $\boldsymbol{P_j} = \langle \boldsymbol{P_{j1}}, \ldots, \boldsymbol{P_{jk}} \rangle$, and $\exists\, \boldsymbol{A_{pj}} \leftrightarrow \boldsymbol{P_j}$, e.g. $\boldsymbol{A_{pj}}$ includes activity-dependent properties for sub-activities.
- $\boldsymbol{S_3}$ = Set of logic flow and controls = $\{\boldsymbol{LG}, \boldsymbol{C}\}$, where $\boldsymbol{LG}$ = logic graph = $\{\boldsymbol{N}, \boldsymbol{A}\}$, and $\boldsymbol{N}$ = Set of activity nodes and $\boldsymbol{A}$ = Set of directed arcs connecting the nodes. $\boldsymbol{LG}$ defines a logical layout of the simulation model that embeds the physical layout of a system, in other words, $\boldsymbol{LG}$ shows all activities required by a simulation model, including those having physical correspondence in real world, and their logical relations. We use a set $\boldsymbol{C}$ to explicitly represent the set of controls required by the application; $\boldsymbol{C} = \{\boldsymbol{C_1}, \ldots, \boldsymbol{C_k}\}$, and $\exists\, \boldsymbol{Ac} \leftrightarrow \boldsymbol{C}$. $\forall j$, $1 \le j \le k$, $\boldsymbol{C_j} = \{C_{jl}, C_{jt}, C_{jr}\}$, where C_{jl} = Control location: An integer represents the sequence number of the last activity node before the control; C_{jt} = Control type: A code represents the type of control, e.g. C_{jt} = {SE = Sequential, RS = Random sampling, SC = Special condition}; and C_{jr} = Control rules: specify the conditions and actions.
- $\boldsymbol{S_4}$ = Set of transfer requirements = $\{\boldsymbol{TR}, \boldsymbol{DS}\}$. Set $\boldsymbol{TR}$ defines a set of entity transfers between activities: $\boldsymbol{TR} = \{T_{ij}, \forall\, ij \in \boldsymbol{A_{TR}}$ = a set of arcs that requires entity transfer$\}$ and $\boldsymbol{DS}$ = A two-dimensional array (i.e. a $n \times n$ matrix, where $n = |\boldsymbol{TR}|$) that captures the physical distances or segments between the points of entity transfers (i.e. processing or servicing activity nodes).
- $\boldsymbol{S_5}$ = Set of system states and control variables, and other parameters used to implement control logic and other system level execution; $\boldsymbol{V} = \{V_1, \ldots, V_p\}$, $\exists\, \boldsymbol{A_V} \leftrightarrow \boldsymbol{V}$, and $\boldsymbol{A_V}$ Includes: (variable) name, (data) type, usage (code), etc.
- $\boldsymbol{S_6}$ = Set of goals/objectives or performance measures specifications; $\boldsymbol{S_6} = \{\boldsymbol{G}, \boldsymbol{M}\}$, where $\boldsymbol{G}$ = Set of codes for goals/objectives = $\{G_1, \ldots, G_k\}$ and $\boldsymbol{M}$ = Set of codes for performance measures = $\{M_1, \ldots, M_h\}$.
- $\boldsymbol{S_7}$ = Experimental design specifications = $\{ES_1, \ldots, ES_m\}$, where the elements can be set as ES_1 = Type of the experiment; ES_2 = Replication design parameters = {Length, Number of replications}; ES_3 = Factorial design parameters = {Factors, Treatment combinations}; ES_4 = Statistics under study or response variables; ES_5 = Confidence level specification, and so on.
- Other **special specifications** (e.g. application domain specific requirements)

5 RULES AND ALGORITHMS TO SPECIFY LOGICAL ACTIVITIES

To determine the logical activities required by a simulation model, we need modeling knowledge to transform concepts from application to simulation domain. This type of knowledge is represented in the form of logic rules, i.e. a set of rules $\boldsymbol{Rs}$ to determine the type and the number of instances of the activities required by a conceptual model. These rules have a generic form:

$$s = \{\ \forall j, \text{ If } \boldsymbol{X_j} \text{ Then } \boldsymbol{Y_j} \text{ Else } \ldots; \text{ or } \boldsymbol{X_j} \Rightarrow \boldsymbol{Y_j}\}$$

Where ***X*** is a conjunction of antecedents (premises or conditions) and ***Y*** is a conjunction or disjunction of consequents (decisions about the activities). Some of the rules are listed below as examples:

- If (More than one entity type share the same arrival pattern)Then (Recommend a single CreateActivity for this group of entity types) Else (Add one CreateActivity for each entity type);
- If (Function(Activity) == "Process") Then (Add a ProcessActivity);
- If (Function(Activity) == "Hold entity temporarily") Then (Add a HoldActivity);
- If (TransferType(After an activity *i*) ≠ "Instant connect") Then (Add a TransportActivity(*i*));
- If (Entities need to be batched before an activity *i*)
- Then (Add an AggregateActivity(*i*));
- If (A batched entity needs to be split after an activeity *i*) Then (Add a DisaggregateActivity(*i*));
- If (PossibleRoutesAfter(*j*) > 1) ∧ (RoutingType = = *k*) Then (Add a BranchActivity of type *k* after activity *j*);
- If (An activity *i* is the last activity in a process plan) Then (Add a DisposeActivity after *i*);
- If (A ProcessActivity P is shared by more than one entity type) Then (Recommend P.ProcessTime = Variable) ∧ (NeedAssignment = True); //Assign each entity type an attribute for "Processing time"
- If (Goal(*j*) = "Estimate flow time" for entity type *k*) then (NeedAssignment = True) ∧ (AssignType = "Arrival time") ∧ (Entity type = *k*) ∧ (NeedCollectData = True) ∧ (CollectType = Time-interval);
- If (NeedAssignment == True) ∧ (AssignType = "*j*") ∧ (Entity type == *k*) Then (Add an AssignActivity (*j*) after the creation of entity *k*);
- If (NeedCollectData ∧ CollectType == "*j*") Then (Add a CollectActivity(*j*));
- If (NeedTransfer(After *j*) == True)∧(Destination == *k*) then (Mark the Arc(*jk*));

As we mentioned earlier, conceptually a simulation model can be defined through a graphical specification, i.e. a graphical representation that shows the logic flow of simulated entities. Several researchers have proposed variants of activity cycle diagram (ACD) based representation (Kienbaum 1994). Like a textual specification, a graphical specification has been used primarily to facilitate user level communication. We enhance this representation through a knowledge-based approach to construct and validate the logic graphs utilizing the knowledge that has been or can be captured and represented. We define a logic graph ***LG*** = {***N***, ***A***}. Every node $j \in \boldsymbol{N}$ is a logical activity required by the model, and it is an instance of some activity type defined in $\boldsymbol{S_2}$. Every arc $l \in \boldsymbol{A}$ connects two nodes and represents a precedence relationship between the two nodes. A logic graph ***LG*** looks like an inverted tree starting with a dummy root node *S*, the "children" of *S* are a set of "Create" activity nodes. A "leaf" node (a node that has no child) corresponds to a "Dispose" activity. Unlike a pure hierarchical tree, loops are allowed in the ***LG*** to accommodate the situation such as sharing of resource and assembling activities. Figure 3 shows part of a logic graph. In the figure, *S* is a dummy starting node (with dashed arcs emanating from it), nodes 1 and 2 are the nodes of Create type, while nodes 8 and 9 are Dispose type. All other intermediate nodes are of certain activity type.

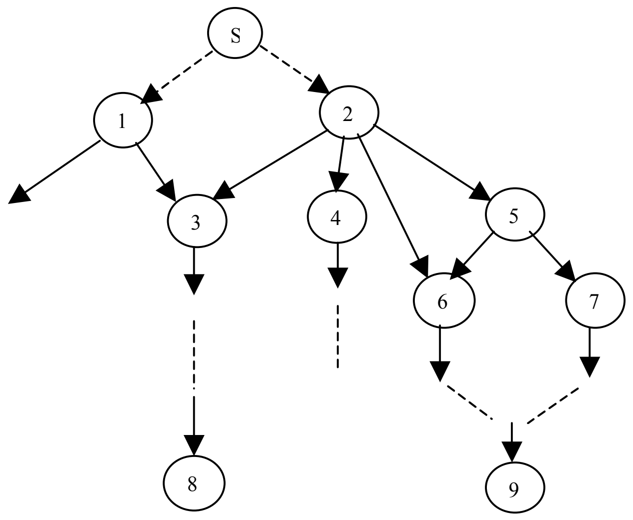

Figure 3: A Logic Graph (*LG*) for Conceptual Model

In the following we propose two algorithms: ***Generate_CSMLG*** and ***Validae_CSMLG***. ***Generate_CSMLG*** interacts with a user to construct a logic graph based on the partial information captured and structured by the notation presented earlier. The execution of this procedure is interactive and iterative. It first tries to establish a logic flow with the activities having physical correspondence (e.g. processing activities); then modify it by inserting additional activities required by simulation (e.g. assigning attributes or collecting statistics). Procedure ***Validae_CSMLG*** automatically validatea the generated graph, e.g. check the consistency and completeness between the graphical entities (nodes and arcs) and their definitions or implications established in a knowledge base (KB). These algorithms have been essentially modified from a depth-first enumeration procedure (Ginsberg, 1993) by addressing the unique requirements of logic configuration for simulation models. The pseudo-codes are shown below:

Algorithm *Generate_CSMLG:*

Initialize $\boldsymbol{\Sigma}$; // $\boldsymbol{\Sigma}$ = a list of initial activities, e.g. Create activities;

```
If ($\Sigma = \Phi$) then return failure;
Set $k$ = Front ($\Sigma$);      //Set $k$ = 1st activity in the list $\Sigma$;
While ($\Sigma \neq \Phi$)
{
    Current_Node = $k$;
    Add_Node ($k$);   //Add a node for activity $k$;
    If (Function ($k$) == "Dispose")
        Then Backtrack ($k$);
    Else {
        Update_List ($\Sigma$);  //$\Sigma = \Sigma \setminus k$;
        Insert_Children (k);
        }
} //end of while loop;
Add_Missing_Arcs( );
```

Agorithm *Validate_CSMLG* *(Given a constructed CSM configuration graph):*

```
Initialize $\Omega$,   //Set $\Omega$ = list of initial nodes (i.e. set of
                    //CreateActivitiy nodes);
If ( $\Omega$ = = $\Phi$) then return failure
Set $j$ = Front ($\Omega$);     //Set node $j$ = first node in the
                          //list $\Omega$;
While ($\Omega \neq \Phi$)        //while set $\Omega$ is not empty;
{
    Remove_Front($\Omega$);    //Set current node to be $j$;
    CheckNode ($j$);          //Call a procedure to
                            //validate node $j$;
    If ((Function(j) == "Dispose") $\vee$ (Checked(j) ==
    True))
        Then BackTrack (j) ;        //Backtrack
                                // from node j;
    Else
    {
        Update_List($j$);   //Remove node $j$ from $\Omega$:
                         //$\Omega = \Omega \setminus j$;
        Insert_Children($j$);     //Add to the front of $\Omega$
                                //all of $j$'s children;
    }
}          //End of the while loop.
```

Where procedure *CheckNode(j)* is given as follows:

```
Procedure CheckNode (NodeID)
{
    For each attribute $A_{2j} \in \boldsymbol{A_2}$
        {
            Initialize_Attributes (NodeID);
            Check_Attribute (NodeID, $A_{2j}$);
            If Valid(NodeID ) == False
                Then present an error message;
            Else Checked (This node) = True;
        }
}
```

For Procedure ***Generate_CSMLG***, *Insert_Children (k)* has several functions: it must identify and insert the set of activities that are needed and qualify to be the "Children" of k and establish the ***links*** between k and its children (determining entity transfers); it must exclude those that have been added, and if a potential child is an added Dispose activity, it will return 0 and backtrack from the current node. In ***Validate_CSMLG***, *Insert_Children(j)* has two main functions: identify the set of nodes that qualify to be the "Children" of node j; and add the children nodes to the "front" of existing list $\boldsymbol{\Omega}$. Similarly it will exclude those that have been checked and return 0 for backtrack.

For this we need to define the concept of "child" and the condition under which a node can be identified as a child. Let n be a parent node (a node that is the root of a sub-tree $T(n)$), and $L(n)$ = a list of nodes $\in T(n)$ that are directly linked to n by the arcs emanating from n. Further let a node $k \in L(n)$, and $C(n)$ = the list of children of n = set of nodes that are uniquely linked (only one path) to n by the arcs emanating from n. Apparently, $C(n) \subseteq L(n)$, i.e. $C(n)$ is a subset of $L(n)$. We make the following proposition:

> **Proposition**: $\forall k \in L(n)$, $k \in C(n)$ if and only if $Parent(k) \cap L(n) = \Phi$, i.e. none of the parents of k is in $L(n)$, or there is only one path from n to k: $(n \rightarrow k)$. (Note that in CSM logic graph, it is possible for a node k to have more than one parent nodes)

Proof: Assume $k \in C(n)$, this $\Rightarrow n \in Parent(k)$, also $n \cap L(n) = \Phi$; let $x \in Parent(k) \Rightarrow$ exist a path $(x \rightarrow k)$; If $x \cap L(n) \neq \Phi \Rightarrow x \in L(n)$ and exist another path $(n \rightarrow x \rightarrow k)$; this contradicts to the assumption $k \in C(n)$. Now assume $n \in Parent(k)$ and $Parent(k) \cap L(n) = \Phi$; this $\Rightarrow$ no $x \in L(n)$ can directly connect k, and the only possible path from n to k is $(n \rightarrow k)$, therefore $k \in C(n)$ ■

To further illustrate this proposition, we give some examples from Figure 3. First we look at node 2, $L(2) = \{3, 4, 5, 6\}$ and $C(2) = \{3, 4, 5\}$. Since $Parent(6) = \{2, 5\}$, i.e. $Parent(6) \cap L(2) = \{5\} \neq \Phi$, therefore node $6 \notin C(2)$. For node 5, $L(5) = C(5) = \{6, 7\}$. Since $Parent(6) = \{2, 5\}$, and $Parent(6) \cap L(5) = \Phi$, so $6 \in C(5)$.

The goal of knowledge representation (KR) is to represent the knowledge in the most appropriate ways for effective and efficient reasoning/inference. KR also depends on the characteristics of the knowledge. Generally there are following types of KR schemes: (1) Semantic networks and frames (Ginsberg 1993): they are easy for understanding; possess good properties such as inheritance, encapsulation and taxonomic structure, and are appropriate for representation of declarative or factual knowledge. (2) Rules (Durkin 1994): infer new knowledge from known information; has a unique role in building cognitive architectures for human reasoning, therefore made it popular in developing practical problem-solver/advisors. (3) Logic (predicate logic) (Ginsberg 1993): more expressive or more flexible.

In addition to be valid, "good" knowledge representations are (1) efficient: facilitate efficient reasoning with the knowledge, e.g. easy to map concepts from the application domain to simulation domain. Classification, generalization, and structured search/inquiry are techniques that can help given appropriate KR scheme; (2) robust: should not be sensitive to the changes of knowledge, or the change/update of the knowledge should be "local". This is good for incremental development and maintenance of a knowledge base; and implementation free, i.e. independent of specific tool selected by user for implementation; and (3) consistent: allow co-existence of different views and multiple levels of abstraction in the same modeling concept or simulation model. This is also called "issue of semantic compatibility" (Zulch 1998).

As described earlier, the knowledge and information contained in a CSM and those used for building a CSM are of two types: declarative or factual knowledge; and procedural or inferential knowledge. For instance, declarative knowledge includes entities, resources, queues, and statistics; while procedural knowledge includes control and routing rules, as well as rules for concept transformation or model construction. We propose the following general strategy for knowledge representation: (1) Use an object-oriented (OO) approach, or a collection of taxonomic structures to model the concepts that are the type of declarative or factual knowledge. These structures possess good properties such as knowledge encapsulation and inheritance, and (2) Use a collection of logic rules to model the concepts that are the type of procedural or inferential knowledge, such as control rules and concept mapping rules. Therefore our knowledge base will generally contain a set of "objects" (a taxonomic structure) and a set of rules that interact with these objects.

6 AN EXAMPLE

In this section, we illustrate the notation and algorithms proposed earlier to construct a logic graph for a simple example. The system manufactures two types of products *A* and *B*, where each has a different arrival pattern. *A* requires a process P1, then an inspection process (P3). *B* requires a process P2, then the same inspection process P3. If the part (*A* or *B*) passes the inspection (90% chance), it goes to a ship-out exit; otherwise to a rework process (P4). After rework, with 80% chance the part becomes useful and is sent to a salvaged-part-exit; otherwise sent to a scrapped-part-exit. Two transporters are used to transfer parts between P1-P3, P2-P3, P3-P4, and from P3 (P4) to corresponding exits. The goals of simulation are to estimate the average flow time and the number of each type of "outputs" (i.e. good parts, salvaged parts and scrapped parts).

For this example, $\boldsymbol{S_1}$ = {e1, e2, r1, r2, r3, r4}, where e1 = A, e2 = B, r1, r2, r3 and r4 are the resources used at processing activity P1, P2, P3 and P4 respectively. Process plans associated with two entities are A_e(e1) = (P1, P3, P4|P3) and A_e(e2) = (P2, P3, P4|P3), where Pj|Pi means that routing to Pj depends on the result of Pi. $\boldsymbol{S_5}$ = {g1, g2}, where g1 = estimated flow time and g2 = number of each type; Control $\boldsymbol{C}$ = {c1, c2}, where c1 = {P3, RS(1, 90%; 2, 10%), c1r} and c2 = {P3, RS(1, 80%; 2, 20%), c2r}. Applying the rules in ***Rs*** we obtained $\boldsymbol{S_2}$ = {Cr1, Cr2, As1, As2, P1, P2, P3, P4, Br1, Br2, R1, R2, R3, D1, D2, D3). Note that $\boldsymbol{S_2}$ includes 2 instances of Create activity (Cr1, Cr2); four instances of Process activity (P1, P2, P3, P4) and so forth. Set ***TR*** = {T_1, ..., T_6}, where T_1 = entity transfer between P1 and P2, and so forth. The graphical model constructed and validated is shown in Figure 4. This logic model communicates clearly about the logic and structure of a conceptual simulation model, and its database representations can then be interfaced with selected implementation tools to create an executable model.

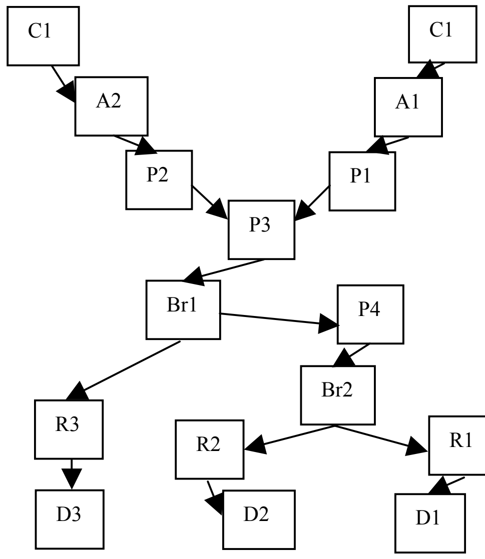

Figure 4: An Example LG Generated by Proposed Rules and Algorithms

7 CONCLUSION

We discussed knowledge-based simulation modeling in this paper, especially the knowledge and its representations for conceptualization modeling. We also presented a notation to formalize the representations to support model development and validation and facilitate the analysis and translation of the representations. We have demonstrated that simulation modeling, especially conceptualization, can be facilitated by identifying and separating knowledge from different domains, and representing them in proper forms, and developing algorithms for structured and automated reasoning. This work fits into a larger project that tries to conduct in-depth study on the knowledge representation and construction of knowledge base for conceptual modeling and facilitate the interface between model conceptualization and implementation.

REFERENCES

Arons, H.D.S. 1999. Knowledge-based modeling of discrete-event simulation systems. In *Proceedings of the 1999 Winter Simulation Conference*, ed. P. A. Farrington, H. B. Nembhard, D. T. Sturrock, and G. W. Evans, 591-597. Institute of Electrical and Electronics Engineers. Piscataway, New Jersey.

Arons, H. D. S. and E. V. Asperen. 2000. Computer Assistance for Model Definition. In *Proceedings of the 2000 Winter Simulation Conference*, ed. J. A. Joines, R. R. Barton, K. Kang and P. A. Fishwick, 399-408. Institute of Electrical and Electronic Engineers. Piscataway. New Jersey.

Banks, J. (editor). 1998. *Handbook of Simulation, Principles, Methodology, Advances, Applications, and Practice*. John Wiley & Sons, Inc. New York.

Durkin, J. 1994. *Expert Systems, Design and Development*. acmillan Publishing Company, New York.

Fox, S. M., N. Husain, M. McRoberts and Y.V. Reddy. 989. Knowledge-Based Simulation: An Artificial Intelligence Approach to System Modeling and Automating the Simulation Life Cycle. In *Knwoledge Based Simulation*, 447-485. John Wiley & Sons.

Ginsberg, M. 1993. *Essentials of Artificial Intelligence*. organ Kaufmann Publishers. San Mateo, CA.

Kienbaum, G. 1994. H-CAD: Hierarchical Activity Cycle iagrams for Object-Oriented Simulation Modeling. *Proceedings of the 1994 Winter Simulation Conference*, ed. J. D. Tew, S. Manivannan, D. A. Sadowski and A. F. Seila, 600-610.

Law, A. M. and W. D. Kelton. 1991. *Simulation Modeling nd Analysis*. 2nd ed. McGraw Hill, New York.

McLean, C. and S. Leong. 2001. The Expanding Role of Simulation in Future Manufacturing. *Proceedings of the 2001 Winter Simulation Conference*, ed. B. A. Peters, J. S. Smith, D. J. Medeiros and M. W. Rohrer, 1478-1486. Institute of Electrical and Electronics Engineers. Piscataway, New Jersey.

Pidd, M. and R. B. Castro. 1998. Hierarchical Modular Modeling in Discrete Simulation. *Proceedings of the 1998 Winter Simulation Conference*. ed. D. J. Medeiros, E. F. Watson, J. S. Carson and M. S. Manivannan, 383-389.

Russell, S. and P. Norvig. 1995. *Artificial Intelligence, A Modern Approach*. Prentice Hall, Upper Saddle River, NJ.

Son, Y. J., R. A. Wysk and A. T. Jones. 2003. Simulation-based shop floor control: formal model, model generation and control interface. *IIE Transactions*, 35:29-48.

Zulch, G., J. Fisher and U. Jonsson. 2000. An Integrated Object Model for Activity Network Based Simulation. *Proceedings of 2000 Winter Simulation Conference*. ed. J. A. Joines, R. R. Barton, K. Kang and P. A. Fishwick, 371-380. Institute of Electrical and Electronic Engineers. Piscataway. New Jersey.

AUTHOR BIOGRAPHIES

MING ZHOU is an associate professor and program coordinator of Mechanical Engineering Technology at Indiana State University. He received a Ph.D. in Systems & Industrial Engineering from The University of Arizona in 1995. Dr. Zhou's research interests include knowledge based simulation and intelligent decision support systems for manufacturing, logistics and distribution systems. He has been a member of IIE and the Editorial Board, International Journal of Industrial Engineering since 1997. His email address is <`imming@isugw.indstate.edu`>.

YOUNG JUN SON is an assistant professor in the Department of SIE at The University of Arizona. Dr. Son received his BS degree in IE with honors from POSTECH in Korea in 1996 and his MS and Ph.D. degrees in IME from Penn State in 1998 and 2000, respectively. His research work involves distributed and hybrid simulation for analysis and control of automated manufacturing system and integrated supply-chain. He is an associate editor of the International Journal of Modeling and Simulation.

ZHI-MING CHEN is a professor and the Associate Dean of the College of Management, Shenzhen University, China. He received a Ph.D. in Engineering from Beijing Areospace University, and had been a visiting professor at the Boston University during 1999 and 2001.