DEGOMS, A SYSTEMATIC WAY FOR TASK MODELLING AND SIMULATION

Ali Mroue[(a)], Jean Caussanel[(b)]

(a)(b)Laboratory of sciences of informations and of systems
LSIS UMR 6168 University of Paul Cezanne, Aix-Marseille III
Av. escadrille de Normandie Niemen 13397 Marseille Cedex 20, France

(a)ali.mroue@lsis.org,, (b) jean.caussanel@lsis.org

ABSTRACT
If the representation of tasks can now rely on many modeling formalisms, few of them offer simulation solution for controlling the consistency of the modeled task. Most often this control can be done by a human agent working on the built model.
As part of a project of modeling and simulation of the human operator, we propose a process and a platform offering an opportunity for those who have to design applications with the known constraints of current developments. This paper describes our approach which uses GOMSL formalism, widely used in the world of HCI, as a model source. GOMSL is very suited for the description of computer task but it doesn't have a real simulator. The aim of this work was therefore to establish a semi-automatic re-expression of GOMSL models to DEVS model that can be immediately simulated (a set named DeGOMS).

Keywords: Task Model Simulation, HCI, DEVS, GDEVS.

1. INTRODUCTION

The Task models in computer science are useful for gathering and organizing the need for the user. They are normally used in the design phase of software to evaluate in advance, test and improve the design of the latter.

Few are the models that describe the real aspect of the user, they are normally based on descriptions of prescribed tasks (Tricot and Nanard 1997), which considers that the user masters the use of application. Even fewer are environments that offer a simulation solution for these task models in order to assess their consistency or their usability.

In addition the improvement of realism of this task models that we will not discuss it in this paper, also the ability to simulate the produced models are considered as an important way for verifying these models.
Our contribution to this field will therefore be as a first step, to design and develop such a platform for testing task models.

We have chose GOMS (Card, Newell and Moran 1983) as a task modeling formalism, which is used for modeling tasks from the view-point of user/system, specially in computer task modeling.
GOMS models can generally be used for the evaluation of interfaces (predicting execution and learning time).
GOMS models are widely used to evaluate and test at a lower cost, interfaces during the design phase. However, GOMS models are designed as models of representation and not as simulation models.
It is therefore necessary to adapt GOMS for simulation or be able to express it in a simulated formalism.

For this reasons we chose the DEVS formalism (Zeigler 1984), which offers a good level of expressiveness, a very good level of scalability, while ensuring formal consistency of the built models.

The equivalent of GOMS model in DEVS (which we called DeGOMS) is a generic level so that any GOMS model can be expressed in DEVS and then simulated. In this theoretical transformation approach, certain choices have been taken which can affect GOMS model (extending it).

2. GOMS MODEL

The acronym GOMS stands for Goals, Operators, Methods, and Selection Rules GOMS (Card, Newell and Moran 1983; John and Kieras 1996). GOMS is a behavior description model, that lets model the behavior at different levels of abstraction, from task level to physical actions.

GOMS uses as a starting point the Model Human Processor principle of rationality that attempts to model and predict user behavior. Its essential contribution is a formal structure that allows organizing the design process.
The design method that induces GOMS is done on two axes:

- In the analysis of task (since determines the behavior).
- In the predictive evaluation of user behavior in the task.

2.1. GOMS Element

GOMS is an approach of modeling human computer interaction.

GOMS models consist of descriptions of the methods required to accomplish a specific goal.

Methods are a sequence of operators and sub-goals to achieve a goal. If there exists more than one method to accomplish a goal then selection rules are used to choose which method to use.

- Goals are tasks the system's user wants to accomplish. For example, "Create Folder, Delete Word". A goal can have a hierarchical structure; this means that the achievement of a goal may require accomplishing one or more sub-goals.
- Operators are actions allowed by the software or actions that user are executing. An operator is an atomic level action that can't be composed, and it's characterized by its execution time. The execution of the operators causes change in the mental state of the users or in the environment state. There are two types of operators mental and physical, For example, the operators “press enter”, “point to the word”, etc. are physical operators. The model also includes mental operators, such as “thinking”, etc.
- Methods refer to the process that allows one to accomplish a goal. Methods are possible sequences of operators and sub-goals used to accomplish a goal. For example the goal of logging into web-mail can be represented as:

    1. Connect_to_the_webmail_provider
    2. Type username
    3. Type password
    4. Press Login

- Selection Rules are used when there exists more than one method that can accomplish the same goal. They are rules used by user to choose which of methods to use.
  A rule has the form:
  If <condition>
  Then use the method M;

GOMS has been used in many applications:
* Telephone operator (CPM-GOMS)
* CAD systems (NGOMSL)
* Text editing using the mouse (KLM)

2.2. GOMS Variation

There are four different models of GOMS: CMN-GOMS, KLM, NGOMSL and CPM-GOMS.

CMN-GOMS stands for Card, Moran and Newell GOMS, is the original version of the GOMS technique in human computer interaction. This technique requires a strict goal-method-operation-selection rules structure.

KLM is a simplified variant of CMN-GOMS, it does not use goals, methods, or selection rules only simple “keystroke-level operators”.

NGOMSL stands for Natural GOMS Language, developed by David Kieras (Kieras 2006). An NGOMSL model is in program form, and provides predictions of operator sequences, execution time, and time required to learn the methods. Like CMN-GOMS, NGOMSL models explicitly represent the goal structure, and thus can represent high-level goals like collaboratively writing a research paper" (John and Kieras 1996).

CPM-GOMS stands for Cognitive, Perceptual, and Motor and the project planning technique Critical Path Method. CPM-GOMS was developed in 1988 by Bonnie John (John and Kieras 1996). CPM-GOMS does not make the assumption that operators are performed serially, and hence it can model the multitasking behavior that can be exhibited by experienced users. The technique is also based directly on the model human processor a simplified model of human response.

In this paper we will introduce the equivalent of the NGOMSL in DEVS. Consequently we will details the NGOMSL of GOMS.

3. NGOMSL

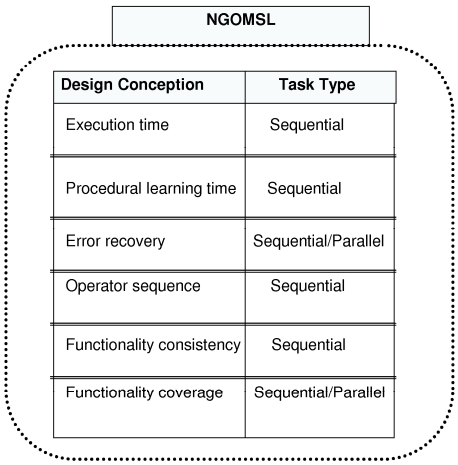

Figure 1: NGOMSL design conception and task type capability

GOMS is a group of models developed by Card (Card, Newell and Moran 1983) and his colleagues at Xerox to predict the time needed to accomplish cognitive activities using a computer system. The model is designed to provide approximations for the duration of the task. The set of GOMS models assume that the user is quite familiar with the task and that the primary human limitations are cognitive (thought), not physiological (aerobic capacity, muscle strength, etc…).

The key of GOMS analysis is the decomposition of the task. As we mentioned above there are four of commonly GOMS implementations concepts CMN-GOMS, KLM, NGOMSL, and CPM-GOMS for more information see (John and Kieras 1996). We chose to work with NGOMSL, since it can be used in most types

of tasks and information design (Kieras 2006) (Figure 1).

NGOMSL is a structured natural language used to represent user methods and selection rules.
NGOMSL models have an explicit representation of the user and his methods, which are supposed to be strictly sequential form and hierarchical. NGOMSL is based on the cognitive modeling of human-machine interaction.
In this paper we present a transformation of NGOMSL to DEVS model which is based on the definition of GOMSL (Kieras 2006) which is an executable representation of NGOMSL.

GOMSL can be treated and executed using a simulation tool named "GLEAN". This tool allows predicting task time and the performance of the user (learning time).
GOMSL syntax is comparable to that of a procedural programming language. As other GOMS models, GOMSL defines the task as of a method composed from steps. Each step is generally composed of one or more operators.

All data in GOMSL are generally represented as objects with properties and values (See LTM_item in Figure 2).

```
LTM_item: Copy_Command
          Name is Copy.
          Containing_Menu is Edit.
          Menu_Item_Label is Copy.
          Accelerator_Key is COMMAND-C.

Method_for_goal: Select Word
          Step 1. Look_for_object_whose Content is Text_selection of
          <current_task> and_store_under   <target>.
          Step 2. Point_to <target>; Delete <target>.
          Step 3. Double_click mouse_button.
          Step 4. Verify "correct text is selected".
          Step 5. Return_with_goal_accomplished.

Method_for_goal: Select Arbitrary_text
          Step 1. Look_for_object_whose  Content is Text_selection_start of
          <current_task> and_store_under <target>.
          Step 2. Point_to <target>.
          Step 3. Hold_down mouse_button.
          Step 4. Look_for_object_whose Content is Text_selection_end of
          <current_task>
          and_store_under <target>.
          Step 5. Point_to <target>; Delete <target>.
          Step 6. Release mouse_button.
          Step 7. Verify "correct text is selected".
          Step 8. Return_with_goal_accomplished.
```

Figure 2: Part of GOMSL methods and object definition

There are several kinds of data used in GOMSL. There is data that represent the knowledge of the users, data that represent the system and data that represent the task list.

The Part that contains data representing the knowledge of user are used and defined in:

Working Memory: Represents the working memory of the user; it has two partitions: Object Store and Tag Store...

- Object Store is a store that contain objects that are currently "in focus" in the user working memory. The Object Store can contains at the same time one Visual Object, one Long Term Memory Item (Object), one Task item (Object) and many Auditory Object with a condition. An auditory object is characterized by a decay time, after it the auditory object will decay.
- Tag store represents the conventional human working memory which can contains values associated with tags (ie: Keyword=Html, the tag is keyword which has HTML as value)

GomsL define many operators that deals with memory (storing value and getting values etc…).

Long Term Memory: Represents the Long Term memory of the user, which contains objects with property values. The content of the LTM is static being specified before the simulation of the model.
The data concerning the information about the system is stored in a memory similar to structure of LTM, and it contains system visual and auditory objects that will be needed by the user to execute task (look at a specific object etc…).

Task memory: This is used to represent the information available to the user about the task.

3.1. Operators

Operators are actions that the user performs. Much kind of operators are defined in GOMSL:

3.1.1. External Operators

They are observable actions used by the user for exchanging information with the system or with other humans. Normally external operators depend of the system and the task. Below is a list of external operators classified by their type, the time and the definition of some of these operators are based on the physical and some of the mental operators used in the Keystroke-Level Model (Kieras 2006).

Manual Operator:

- KeyStrock Keyname (Stroke a key on the keyboard. Estimated execution time 280ms).
- Click mouse_button (clicking the mouse button. Estimated execution time 200 ms).
- Hold_down mouse button (press and continue pressing the mouse button. 100ms)
- Etc…

Visual Operator:

- Look_for_object_whose property is value, … and_store_under <tag> (its a Mental operator that searches on the system for a visual object that whose specified properties have the specified values, and stores its symbolic name in the tag store section of the WM, and put the object in focus in the visual part of store object. After the execution of this operator all the properties of the object become available. Execution time is estimated to 1200 ms.

- Look_at object_name (Means looking at an object that has already been identified with a Look_for… operator etc… 200ms)
- Etc…

Etc…

3.1.2. Mental Operators

Mental operators are user internal action, they are not observable or hypothetical, and they are inferred by the analyst or the theorist.

Some of these operators are built in; they are primitive operators corresponding to the mechanism of cognitive processor. These operators include actions like taking decisions; store a value in tag store, search for information in the Long Term Memory, get information about the Task, etc… Other operators are defined by the analyst in order to represent complex mental activity such as "Verify", "Think Of" …

Memory storage and retrieval:

- Store Value under <tag> (Store a value in the tag store under the label tag).
- Delete <tag> (Delete the value stored under the label tag)
- Etc...

flow of control:

- Accomplish_Goal: Goal_name (Accomplish a goal, the goal will be considered as a sub-goal)
- Decide: Conditional; Conditional; …; else-form (This operator let us taking decision. It contains one or more If-Then conditionals and at most one Else form)
- Etc…

Etc…

4. DEVS FORMALISM AND GDEVS

DEVS (Zeigler 1984) is a formalism used to represent Discrete Event System Specifications; it can represent complex system in an effective way. DEVS model is a powerful simulation model. It is a modular formalism for deterministic and causal systems. It allows for component-based design of complex systems. Several specific platforms for DEVS models simulation can be found.

A DEVS model may contain two kinds of DEVS components: Atomic DEVS and Coupled DEVS. An Atomic DEVS does not contain any component in it. It only has a mathematical specification of its behavior. A Coupled DEVS is a modular composition of one or more Atomic and Coupled DEVS.

An atomic DEVS model has the following structure:

$D = <XD, YD, SD, \delta extD, \delta intD , \lambda D, taD>$

Where:

XD: is the set of the input ports and values

YD: is the set of the output ports and values

SD: the sequential state set

$\delta extD: Q \times X \rightarrow S$, is the external transition function where $Q = (s, e) \rightarrow s \in S, 0 \leq e \leq ta(s)$ is the total state set.

$\delta intD: S \rightarrow S$, is the internal transition function

$\lambda D: S \rightarrow Y$, is the output function

$taD: S \rightarrow R+0, \infty$ (non-negative real), is the time advance function.

A coupled model, also called network of models, has the following structure:

$N = <X, Y, D, \{Md / d \in D\}, EIC, EOC, IC>$

X and Y definitions are identical to XD and YD of an atomic model. The inputs and outputs are made up of ports. Each port can take values and has its own field of values.

$X = \{ (\rho,\mu) / \rho \in IPorts, \mu \in X\rho \}$

$Y = \{ (\rho,\mu) / \rho \in OPorts, \mu \in Y\rho \}$

D is the set of the model names involved in the coupled model.

Md is a DEVS model. The variables representing the inputs and the outputs of the model will be indexed by the model identifier. Hence the following notation:

$Md = <Xd, Yd, Sd, \delta extd, \delta intd, \lambda d, tad>$

The inputs and the outputs of the coupled model are connected to the inputs and outputs of the included models.

$EIC = \{((N, a), (d, b))/a \in IPorts, b \in IPortsd\}$

The set of the coupled model input ports ipN associated with the input ports ipd of the models D are the components of the coupled model.

There is the same situation for the output ports.

$EOC = \{((N, a), (d, b))/a \in OPorts, b \in OPortsd\}$

Inside the coupled model, the outputs of a model can be coupled with the inputs of the other models. An output of a Model cannot be coupled with one of its inputs.

$IC = \{((i, a), (j, b))/i,j \in D, i, j,a \in OPortsi, b \in IPorts j \}$

GDEVS is an acronym for"Generalized Discrete Event System Specification" (Giambiasi, Escude and Ghosh 2001) which is a model which generalizes the concept of discrete event modeling. GDEVS defines abstraction of signal with piece wise polynomial trajectory. Thus, GDEVS defines event as a list of values. DEVS can be considered as a particular case of GDEVS, other saying DEVS is an order 0 GDEVS model.

5. GOMSL TO DEVS

GOMSL is a model of representation that is intended to be interpreted and verified by specialists in the field.

It lies at a high level of abstraction to facilitate its interpretation by humans but don't let us testing automatically the produced models.

We believe there is an interest to propose a simulation of represented tasks in order to obtain results or to ensure consistency, in particular when they have reached a certain level of complication.

The choice of target formalism in which the GOMSL representation will be translated is not critical

as long as it ensures consistency syntactical and semantic of constructed models and let them be simulated directly.
For the reason of type of models, our choice fell on Discrete EVent System formalism DEVS simulation (Zeigler 1984; Giambiasi, Escude and Ghosh 2001).

In addition to the characteristics listed above, DEVS is generally having a very good level of scalability and expressiveness (Zeigler 1984).

In addition, we have now in our laboratory a DEVS simulation platform (lsisDME) through which we could test our results immediately.

5.1. Transformation

A GomsL model is equivalent to a coupled DEVS model.

During the transformation we separate the GOMS Methods from other models. So we create a model that represents the user, a model that represents the task, another that represents the system and final one that represents GOMS methods.

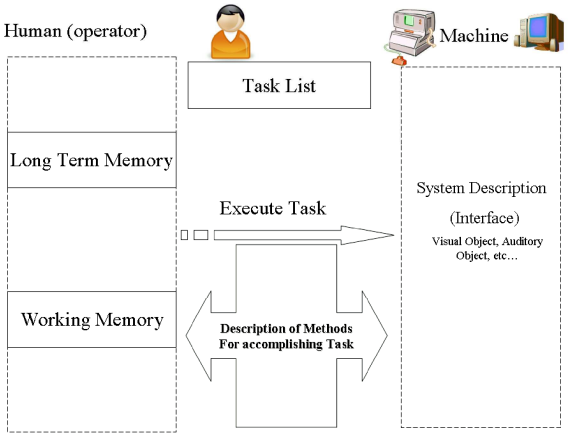

Figure 3: Decomposition of GOMS model in DEVS

This decomposition of model let us obtaining a modular architecture which will be useful so any change in any model will not affect other models moreover any model can be used alone (Figure 3).

The start point is the model representing Methods and Selection Rules. Each method is composed from a sequence of steps. Each step is composed from one or more operators. These operators can communicate with the other different models, in order to get value from memory or save value etc…

In this paper we will describe in detail working memory module used in GOMSL and its equivalent in DEVS (Figure 4). We will discuss then the description of a part of the module that represents “methods and selection rules”.

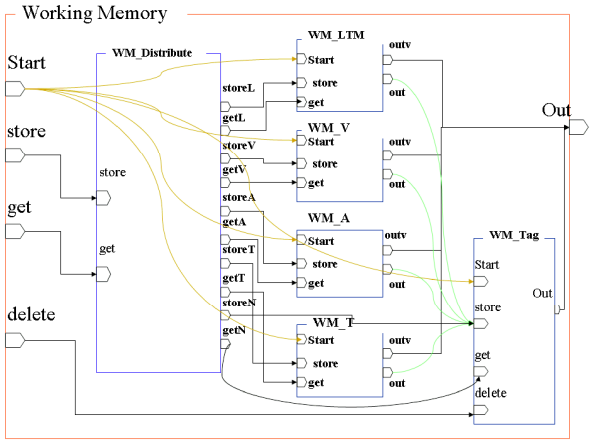

Figure 4: Working Memory GDEVS representation

5.1.1. Working Memory Model

Expressed according to the GDEVS formalism, the "Working Memory" can be defined as follows:

WM: <X, Y, M, EIC, EOC, IC, select> {
X = {Start[0], store[2], get[2], delete[0]};
Y = {Out[1]};
M = {WM_Distribute, WM_LTM, WM_V, WM_A, WM_T, WM_Tag};
EIC = {{Start, WM_LTM.Start}, {Start, WM_V.Start}, {Start, WM_A.Start}, {Start, WM_T.Start}, {Start, WM_Tag.Start},
{store,WM_Distribute.store},{get,WM_Distribute.get}
};
EOC = {{WM_LTM.outv, out}, {WM_V.outv, out}, {WM_A.outv, out}, {WM_T.outv, out},{WM_Tag.out, out}};
IC = {{WM Distribute.storeL, WM LTM.store},
{WM Distribute.getL, WM LTM.get},
{WM Distribute.storeV, WM V.store}, {WM Distribute.getV,WM V.get}, Etc. . . };

The Working Memory module is equivalent to a GDEVS coupled model, which is composed from six models: WM_Distribute, WM_LTM, WM_V, WM_A, WM_T, WM_Tag. It has 4 inputs ports (Start, Store, Get, Delete) and only one output port (Out).

WM_Distribute has as role to distribute every request to the proper model. For example, if WM_Distribute receives a request for retrieving a value from the visual memory part, so the WM_Distribute redirects the request to the proper model which is WM_V. This latter will receive the request and searches the value and sends it to the output.

Models "WM_LTM", "WM_V", "WM_T" are GDEVS atomic models that have a similar structure. Each of them represents a specific part of the Working

Memory, since, according to GOMSL, the working memory may contain various objects at the same time in different zone (visual area, LTM area, Task area). And each zone may contain a single object at the same time (when an object is present in the memory, all his properties become accessible in the methods).

An object in GOMSL is characterized by a name, and by properties with certain values. In DEVS an object is represented by a phase in which are defined state variables having values (These state variables represent the properties of the object).

As seen in the preceding parts, GOMSL defines that the working memory may contain at the same time a visual object, an object from the LTM and a Task object. By respecting this definition, we have created the models (WM_V, WM_LTM, WM_T).

Each of these models is equivalent to an atomic GDEVS model with the following structure:

WM_V = <Xd, Yd, Sd, δextd, δintd, λd, tad>

X = {Start[0], store[1], get[1]};

Y = {out[2],outv[0],message[0]} ;

There are five major states which are (init, receive, select, get, clear) and "n" other states. "n" is equal to the number of objects be stored in memory (Figure 5).

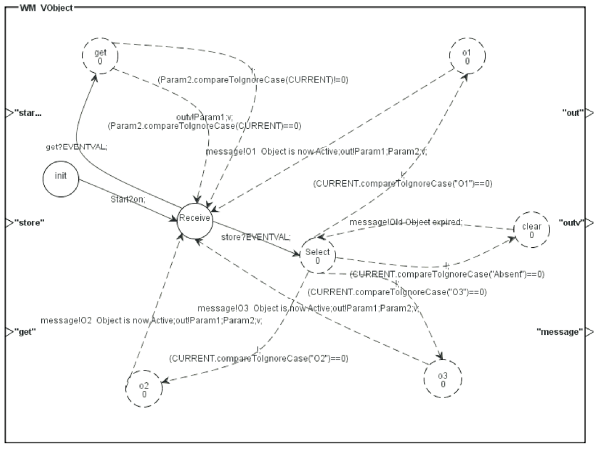

Figure 5: Example of WM_V model

The init phase: Is the initial phase of the component model. It contains the initialization all state variables used in the model.

The Receive phase: This phase can receive two types of requests (storage and getting requests), and according to them it will go to the next phase. For example, if it receives a request to store a value the next phase will be the "Select" phase; else if it receives a get request next phase will be the "get" phase.

The Select Phase: This phase is used to select the memory object to be stored. In other words, after receiving a store request with a tag name and an object name. The select phase will select the object and make it active, what makes all its properties available to future get requests.

For example, if we receives a store (Tag1; O1), then we will go from the Select phase to the phase representing O1, after this the system will send to the output port "out" the values (tag;O1;v), which will be received by the WM_Tag in order save O1 in Tag …. Then the system returns to the receive phase and wait new request.

The Get Phase: This phase will search for a property value and sends it to the "outv" output port.

The clear phase: The role of this phase is to reset the values of all state variables. It will be used when we want to store a new object so the old object must be removed (all its properties must be cleared).

As we mentioned above the n other states represent the object in memory. When the system goes to one of these phases, all the state variables of this phase will be affected with values that represents properties values. After this the object will be active and all its properties will be available for get requests.

The models WM_LTM and WM_T have the same structure of the WM_V model, but n in these cases will be the number of LTM objects or TASK objects.

The WM_A model is different from the other models for the reason that working memory can contains many auditory objects in the same time, but in this case decay time will be taken in consideration. So object after a decay time will automatically be removed.

The WM_A is equivalent to a GDEVS coupled model, composed from n model, where n is the number of auditory objects.

Every auditory object is equivalent to an atomic GDEVS model with the following structure:

WM_V = <Xd, Yd, Sd, δextd, δintd, λd, tad>

X = {Start[0], store[1], get[1]};

Y = {out[2],outv[0],message[0]} ;

This model is composed of 6 phases (Figure 6) which are (init, get, receive, select, get1, get and a phase representing the object).

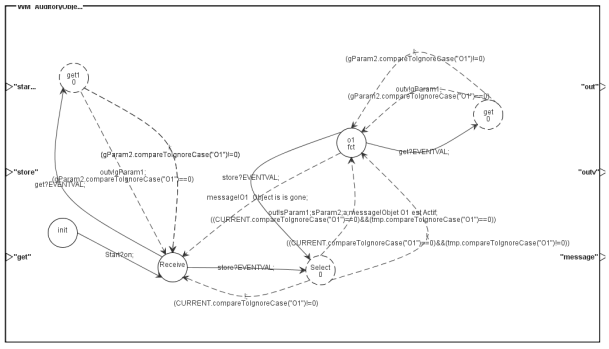

Figure 6: Example of WM_A model

The phases are similar to the ones used in WM_V model, with minor difference in the transition conditions and lifetime function. In other words the phase representing the object "O1" has a lifetime equal to the decay time of the object, after this time all the properties will not be accessible.

The model "WM_Tag" is an atomic GDEVS model representing the "Tag Store" part of the Working Memory.

Each object in the memory must be linked to a tag in the "Tag Store". So we can access objects by using tags. For this reason we can find in the working memory module that all object models are connected to the WM_Tag. As we mentioned above the WM_Tag (Figure 7) is equivalent to an atomic GDEVS model composed of five phases (init, tag, tag_store, tag_delete, tag_get).

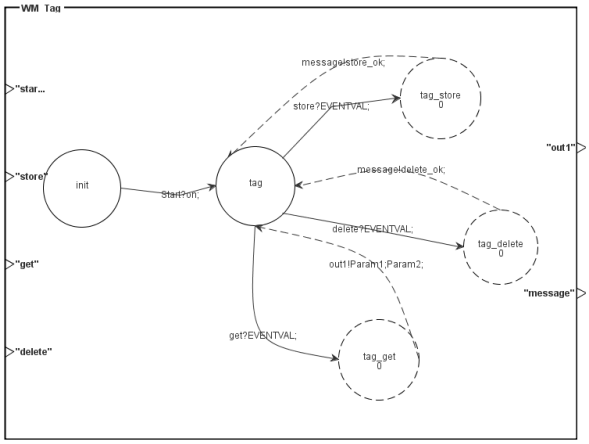

Figure 7: Example of WM_Tag model

The init phase is the initial phase; it contains the definition of all the tags used in the task as a state variables. In other words this phase is used to initialise the values of the state variables.
The tag phase is a waiting phase which will wait for an input event. This phase can deal with three type of events (delete, store, get).

The tag_store phase is used to store a value in a given tag. In other words, after receiving in the store input port a value such: store: current_task;aaa;n; this will be equivalent to current_task=aaa, and current_task_type=n; so in the phase tag_store we will assign value to the state variable.

The tag_get phase is used to retrieve the value from a variable (tag).
The tag_delete phase is used to remove the values from a variable (tag).
The input ports are X= {Start [0], store [2], get [0]};
The output ports are Y = {out [1], message [0]};

The definition of the input and the output port is similar to the one used in the above model.

More generally the working memory module receives at the start of simulation in the port Start an activation signal (on), which puts all the components of the memory in a waiting phase. Then it can either receive requests for storage ("store" input port) or retrieving data (“get” input port) or requests to delete a tag in the WM_tag ("delete" input port).

For example: if the WM receives Store: (query; html; t), the "WM_distribute" analyses all the entries and chooses model to activate depending on the type of application (in this case is part WM_T indicated by the “t” in the input) and then the model will send (query; html) to the port StoreT.

Then, the WM_T model receives on its input port store the values (query; html). So the WM_T will search for the object named HTML and makes it 'active' (all properties of this object will be available...). After that, WM_T sends the values (query; html; t) to the output port "out". Finally, WM_Tag receives on the input port values (query; html; t) and stores in the tag (state variable) name "query" the value "html", and stores also in the state variable “query_type” the value “t” which means "task object”.

5.1.2. GOMS Module

This module represents in general goal, operators, methods and selection rules.

The "Goal" is equivalent to a GDEVS coupled model, composed from a “Selection Rules” model and methods models. The name of the model is equivalent to the goal name.

The model have 5 input ports (Start, memin, taskin, systemin, inaccomplish) and 8 output ports (memstore, memget, memdelete, taskcmd, systemget, LTMget, outaccomplish, accomplished).

Input Ports:
Start: used to indicate the start of execution of the model.
Memin: is the port used to receive values from the working memory.
Systemin : This port is used to receive values from the System.
Taskin: used to receive values from the Task description model.
Inaccomplish: is used to receive the end of execution of another goal called from this goal.

Output Ports:
Memstore: used to send request to the WM for storing values or object.
Memget: used to send request to the WM for retrieving values (object properties values, tag values, etc...)
Memdelete: send a delete request to the WM (The WM_tag exactly) for deleting a tag.
Taskcmd: used to send command to the Task description model (modifying the task list, getting values …)
Outaccomplish: used to send request for executing a goal.
Etc.

The simulation starts with the Selections Rules model. This latter is defined in GOMSL in the form of simple conditions (if then .....). The Selection Rules model is equivalent to an atomic GDEVS model. It has two input ports (memin and Start) and two output ports (memget, out). The port "out" in this case is used to transmit the name of the method selected (Figure 8).

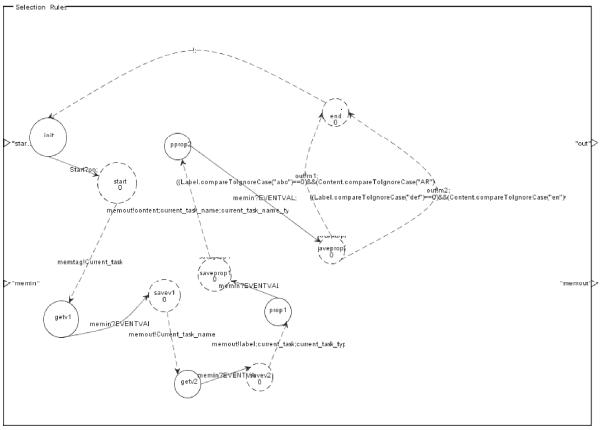

Figure 8: Example of Selection Rules Model

This model is composed of three main phases (init, start and end), plus 2*n phases, with n equal to the number of values we need to retrieve from the "WM". The multiplication by 2 is due to the reason that we have to one phase for requesting values from the WM and another for receiving the value.

First of all this model sends requests to the WM in order to get all needed values, then these values will be compared and according to the result of comparisons the name of proper method to execute will be sent to the output port (out).

The phase that is linked directly to the phase "End" will send the name of the method to execute (according to the condition on transitions. . .)

A Method in GOMSL is equivalent to a coupled GDEVS model composed from n atomic GDEVS model. The "n" Atomic GDEVS model represents the steps of the NGOMSL model.

A Step in GOMSL can be composed from one or more operators.

An Operator in DEVS is equivalent to one or more phases according to the type of operator. For example:
An external operator such as: Click mouse_button is normally equivalent to one phase, the life time of the phase is equal to the execution time of the operator. If this operator needs an access to the memory, then we add 2 other phases in order to send the request to get the value and save it.

Another example for the mental operator such as think is equivalent to one phase etc...

6. DEGOMS ADVANTAGE

The translation of GOMSL into DEVS has extended the features of GOMSL model.

1. To simplify, Gomsl considers that the auditory memory can contain all auditory objects simultaneously. In DeGOMS the contrary, we represent this part of the memory as it is defined, which is to take into consideration the decay time
2. In GomsL, one can not easily add operators. Each added operator needs recompilation and reconfiguration of the existing simulation tools (which is "Glean"). While in DeGOMS, we introduced a generic concept the definition of the operator (example: 1-Request for a value from the memory needs two phases 2-The running time of the operator is equivalent to the life time of the phase, and so on).
3. The modular architecture of DeGOMS allows us to use each module alone, in other applications etc… In the case of modification of a module this will affect only the modified module and not all the modules. In GomsL and Glean this is not possible; since these models are not defined formally (we mostly change the code etc).
4. DeGOMS can be easily expanded by adding models that already exist or creating new models that can be simulated directly. In GomsL/Glean any change is quite complex, and requires coding and recompilation of Glean.
5. Etc.

7. RESULTS

The first results of simulations of DEGOMS models, show the proper functioning of each module taken separately (memory module, method, and so on…)

The simulation of a simple model for the management of files in Macintosh System validates the model and gives good results. It gave us the possibility to obtain the same result of an existing Glean simulator (ex: Accomplishing task time: will be the date of the end of the simulation (Last Event). The advantage of DeGOMS models is that they provide a very large modelling flexibility (easily change models, adding new operators, and adding new methods) also they provide us and the possibility of simulation directly.
This flexibility is not found in an approach as GLEAN (Kieras 2006) which it is a runtime environment model for GOMSL more than a true simulation model.

8. CONCLUSION

The simulation of the operator in its task is challenging, especially when the modeled system includes a human interaction. In the design of computer applications this type of approach allows us to minimize the cost and time of the design while improving the overall ergonomics applications. We designed DeGOMS and developed a DEVS formalism representing GOMSL models and an environment which allows the simulation of produced models.

In this work we succeeded in finding a new simulation tool for the NGOMSL model, which is effective and gives good results. This transformation into DEVS has many advantages, first we have obtained an effective simulation tool for the NGOMSL model, and secondly we can easily interpret these models and extend them in order to add many other modules. And for now we can also do the simulation in interactive mode, and implement the core of the DEVS simulator in a user training applications. The current prospects

relate to model the task of information retrieval on the web and simulating it.

9. REFERENCES

Tricot, A. and Nanard, J. 1997. *Un point sur la modélisation des tâches de recherche d'informations dans le domaine des hypermédias*, in Hypertextes et Hypermedia J.P. Balpe, et al.,Editors. Hermes, 35-56.

Card, S. K.; Newell, A. and Moran, T. P. 1983. *The Psychology of Human-Computer Interaction.* Lawrence Erlbaum Associates, Inc. Mahwah, NJ, USA.

Kieras, D.E. 2006. *A Guide to GOMS Model Usability Evaluation using GOMSL and GLEAN4*. University of Michigan

John B. E. and Kieras D. E. 1996. *The GOMS family of user interface analysis techniques: comparison and contrast.* ACM Transactions on Computer-Human Interaction, 3(4):320–351.

Zeigler B. P. 1984. *Theory of Modelling and Simulation.* Krieger Publishing Co., Inc. Melbourne, FL, USA.

Giambiasi, N. Escude, B. Ghosh, S. 2001. *G-DEVS A Generalized Discrete Event Specification for Accurate Modeling of Dynamic Systems in:* Autonomous Decentralized Systems. Proceedings. 5th International Symposium on.