MULTIFACETTED MODELLING IN SYSTEM DYNAMICS

Graña M., F.J. Torrealdea, F.Ferreres, J.J. Dolado
Facultad de informática
University of the Basque Country UPV/EHU

Abstract We propose a functional design for a software system whose aim is to provide support for an structured methodology of modelling and simulation in System Dynamics. The design follows mainly the ideas of Multifacetted Modelling developed by Zeigler . Our approach has been to give a hierarchical version of DYNAMO and a collection of functions for the handling of simulation elements in an unified system.

0-INTRODUCTION

Seeking for a methodological improvement of System Dynamics modelling, we have found that the Multifacetted Modelling ideas developed by Zeigler are the right framework for this search. In (Torrealdea et alters 1986) we discuss some concepts related to hierarchical modelling, that can be relevant for System Dynamics modellers, the present work must be understood as an extension for that one. Here we try to give a first approach to the design of a software tool as a first goal to be attained, in order to introduce Multifacetted ideas into the System Dynamics realm.

Our design principles have been:

a-Providing a hierarchical language as the basic formalism for model formulation

b-Maintenance of multiple models and their relations within a single system.

c-Growing availability of qualitative analysis tools, through modular enrichments of the package.

d-Abstract interface with the user, through functional (command-like) use of it.

Actually point c has not been discussed in this paper, but it is one of our main goals.

Section 1 gives a brief summary of Multifacetted Modelling sources. Section 2 sketches our approach. Section 3 gives a brief review of the proposed hierarchical language. Sections 4,5,6,7 describe the main functions intended for the package at the present state of design.

1-MULTIFACETTED MODELLING: A SUMMARY

In this section we give an informal review of Zeigler's work. We will outline the most remarkable ideas as a conceptual background for our own work.

The motivation behind multifacetted modelling is threefold:

a-Formalization of simulation concepts in terms of mathematical general systems theory, alowing the formal posing and answer of questions such as the ability of a program to implement a model, the model validation problem, valid simplification of models and so.

b-To set the guidelines for simulation software development.

c-Development of a general methodology for system simulation, supported by the suitable software. This methodology intends to be enough general to be useful whatever would be the application field of the simulation user.

The main references for multifacetted modelling are the books of Zeigler (Zeigler1976, 1984a). Besides these, the reader will find in (Zeigler1978) a good sketch and in (Oren & Zeigler 1979) a more user-oriented approach. The incidence of hierarchical modelling concepts into the formalism is given in (Zeigler1984b).

In (Zeigler 1976) Zeigler sets the basis for the formalism used after, whose cornerstone is the hierarchy of system specifications. We believe that it is interesting to reproduce here informally the levels of this hierarchy.

Level 0-Observation frame: Identifies the time frame and Input-Output values for the system observation.

Level 1-I/O Relation Observation: The system is described by a collection of pairs of Input-Output trajectories : as an Input-Output object.

Level 2-I/O Function Observation: Assumed the ability to set an initial state of the system which univoquely determines its response, the Input-Output pairs constitute an Input-Output function.

Level 3-I/O System: At this level we can decompose time trajectories into segments. Also we can deduce by a transition function the final values reached by the state and output variables, when the system recives an input segment in a given initial state.

Level 4-Iterative Specification: The system is specified in such a way that an appropiate interpreter (algorithm) can deduce (simulate) the corresponding I/O System. At this level we find that various specializad formalisms (discrete event, differential equations and sequential machine) can be translated into a basic iterative formalism (so they are simulable).

Level 5-Structured System Specification: The sets and functions involved in the system description can be structured, this means that we can identify and discriminate elemental sets and functions with specific meanins, whose crossproduct produces the sets and functions of lower specification levels.

Level 6-Network of Specifications: The system is described as a network of systems coupled between them.

Three observations are worth to be done:

a-Going up in the hierarchy we get an increasing knowledge of the system internal structure, and conversely, going down we lose structural information.

b-Given a system specified at level i, it is always possible to deduce its specification at lower levels, the converse is false.

c-The usual simulation languages can be placed as specifications at levels 5 or 6 in this hierarchy.

Once the preceding has been stablished, Zeigler examines for each level the existence of relations which preserve the specification structure (morphisms). Also he studies how a morphism stablished in a level i can induce an equivalent morphism in a lower level, and under what conditions a morphism in a higher level can be inferred from that of level 1. This will prove to be a fruitfull foundation for the formal analysis of several simulation concepts. Over this formal background a set of postulates create the searched framework for the formal discussion of simulation concepts. They can be summarized as follows:

P1-There exists a real system R, which is identified as a universe of potentially acquirable data.

P2-The base model B fully characterizes (modelizes) the real system R. The base model is a closed system specification of level 3 (without Input or Output) and actually it is an ideal object, usually not known.

P3-There exists a set of experimental frames restricting experimental access to the real system.

P4-An experimental frame E specifies the Input to the system, the observable Output and the control conditions needed for the experiment definition.

P5-The real system observed within the experimental frame E is structurally characterized by the base model in E : B/E, which is a full specification of level 3.

P6-The data acquirable by observation of the real system within the experimental frame E are identified with the I/O Relation Observation of the base model in E : $R_{B/E}$ (which theoretically could be formally deduced from B/E).

P7-The real system R is the set of data acquirable throgh any of the experimental frames given for it : the union of all $R_{B/E}$

P8-A lumped model is an iterative system specification M (level 4) . S(M) is the I/O System (level 3) deduced from M by its interpretation.

P9-A lumped model M is valid for the real system in experimental frame E if its I/O relations (level 2) are equal : $R_{S(M)} = R_{B/E}$.

P10-A program is an iterative system specification P.

P11-A program P is a valid simulator of a lumped model M if there is a specification morphism from P to M.

Once created this framework, a number of typical simulation issues are discussed formally by Zeigler, among them we detach the validation and simplification/elaboration issues.

Validity implies the existence of a behavioral equivalence $R_{S(M)} = R_{B/E}$ from which some kind of identity between the base and lumped models can be justifiably accepted. So validity is concerned with the comparison of data from the real system and lumped model, and this comparison can be done in several ways : pure equality, statistical tests and other techniques stablishing a measure for the data equivalence. Validity says nothing about the question of the degree with which the lumped model has apprehended the structure of the base model (real system). To address this question we must start from the validated model and try to set the conditions under which the behavioral equivalence implies structural relations (homomorphisms) between the base and lumped model. This has been called "structural inference".

Simplification/elaboration of models implies the existence of homomorphic relations between models. Particularly, the base model is conceptualized as the most elaborate model we can think of for the system, so that every conceivable model will be a simplification in some way of the base model. Every model is thought as an homomorphic image of the base model. In the same way that the validity of a lumped model with regard to the base model was discussed, the validity of a simplification for a given model can be examined and, further than this, if we can assure that the simpler model is an homomorphic image of the more complex one, then the former will share the validity of the later. Simplification/elaboration relations (taken as homomorphic relations) structure the set of models produced during a simulation project, and play an special role in the modelling process.

In the usual case, where the base model is utopic, the experimental frames are defined in reference to each model, and a relation "is appilcable to" is provided. This relation determines whether a given frame specifies admisble input, output and control conditions for a given model. Also, the set of possible experimental frames will be structured by a "derivability" relation which stablishes a partial ordering in it : An experimental frame E' wil be derivable from another E if the data acquirable in E' is more constrained than that of E. As a consequence the data in E' can be deduced from the data in E.

Multifacetted modelling, methodology and software development, deals with the handling of the evolving environment the modeller builds along the modelling process. This environment is defined by three elements or dimensions:

E1-A collection of experimental frames, structured by the derivability relation. Each frame being applicable to a number of models and the real system. Experimental frames can be conceptualized as questions posed to the model or real system.

E2-The real system, represented by the data that has been collected by experimentation with the real system under certain experimental frames. Also some structural information can be used to represent the more cualitative and abstract descriptions of the system.

E3-The set of models which will be mainly structured by the simplification/elaboration relations. Each model will be valid or not under certain experimental frames.

During the modelling process, new experimental frames, new real system data or new model formulations can be added to the existing environment, which can mean to restructure one or more of the preceding sets. One of the major requirements for a proper multifacetted modelling software will be automated integration of new experimental frames, data or models in the modeller's environment. Other requirements could be multiple formalism for the model specification, intelligent aid in the simplification/elaboration process and friendly interface for the creation of experimental frames, models and data.

2-OUR APPROACH

In this section we give the main lines of our attempt to bring Multifacetted Modelling ideas into a software system for System Dynamics modelling. At the moment of writing this, we have not yet begun its realization, so the following must be seen as a proposal rather than a description. Next sections will give more detailed account for the points outlined now. All the examples will be worked on the models discussed in Alfed & Graham (1976).

Let us call, for the sake of clear reference, MULTI to our software system. So, MULTI will try to deal with the modeller's environment, giving him a unified interface to manipulate the objects actually in this environment and to structure them. Also MULTI must be a feasible thing, so it will show many restrictions and simplifications relative to an ideal "good" multifacetted software system.

To be precise, we will consider the following sets of things as the modeller's environment:

-A Set of Models, where each model will be a set of first-order differential equations, specified in a language near to DYNAMO

-Real System Data. A set of time trajectories obtained from the real world observation under one or more experimental conditions.

-Model Generated Data. A set of time trajectories given by the model simulation under certain experimental conditions

-A Set of Experimental Conditions, which are the Multifacetted Modelling experimental frames, but applied to System Dynamics models.

-A set of cualitative informations, under the generic name of Documentation, relative to models, real system and experimental conditions.

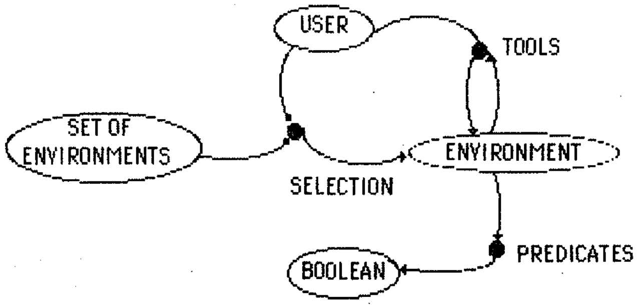

figure 1 Basic functionality of MULTI

MULTI will provide for this ENVIRONMENT representation and storing. Also, MULTI will provide a collection of functions for the environment manipulation, which will be called TOOLS. Also, a collection of tests will be provided to verify that certain conditions are accomplished by the environment elements, whose generic name will be

PREDICATES. An special activity of the modeller is that of environment SELECTION. Under MULTI more than one environment can be created, but it is unfeasible to work with more than one each time. So the first action of the user must be that of environment selection. After that, he can apply any of the provided TOOLS to change the environment state. Figure 1 shows a functional digraph for the preceding.

We wil use (informally) through the text a functional notation inspired by the algebraic method for software specification. (The motivated reader is addressed to (Liskov & Zilles 1975) for a survey). So, TOOLS, SELECTION and PREDICATES must be interpreted as abstract functions mapping the abstract sets USER, SET_OF_ENVIRONMENTS and ENVIRONMENT, and the more concrete set of BOOLEAN values. So we can write:

SELECTION : USER x SET_OF_ENVIRONMENTS → ENVIRONMENT
TOOLS : USER x ENVIRONMENT → ENVIRONMENT
PREDICATES : ENVIRONMENT → BOOLEAN

Where ENVIRONMENT is the set of all possible environments and SET_OF_ENVIRONMENTS is the set of all possible sets of environments. USER will be used to specify the set of possible interactions of the modeller with MULTI; its abstract elements will range from numbers or names (in TOOLS or SELECTION definition) to classes of text specifications.

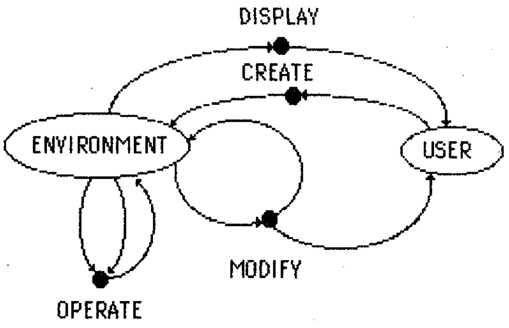

figura 2 TOOLS classes

TOOLS actually is not an abstract function, but a collection of them. In a first approximation we can distinguish four function classes within it. Figure 2 illustrates these classes as abstract functions in themselves. Each class, expressed as an abstract function, precises the basic pattern of its members.

DISPLAY : ENVIRONMENT → USER
CREATE : USER x ENVIRONMENT → ENVIRONMENT
MODIFY : USER x ENVIRONMENT → ENVIRONMENT
OPERATE : ENVIRONMENT x ENVIRONMENT → ENVIRONMENT

3-THE BASIC FORMALISM

The models in MULTI will be expressed in a language which can be regarded as a hierarchical extension for the common System Dynamics language DYNAMO. The description given here will be short, for more detail and discussion of the concepts involved, the reader is addressed to Torrealdea et alt. (1986)

The main feature of the language (let us call it H-DYN provisorily) is the recursive nesting of coupled models. The general pattern of specification is as follows :

Model Heading
  Specification of Component Models
Model Equations

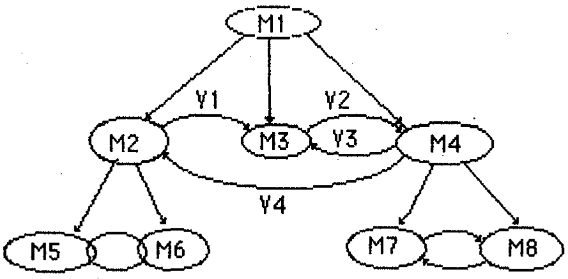

figure 3 Example of model structure

DISPLAY is the class of abstract functions that will serve the modeller to observe the environment state. Here USER means a set of graphical or textual representations. Through DISPLAY functions, the modeller will observe for example the actual specification of a model or a set of time series.

CREATE is the class of abstract functions that will serve to introduce new elemnts (models, experimental conditions, data or documentation) into the environment. They will be associated with compiling facilities, given that USER means a set of texts (in our simple, prototypical design)

MODIFY is the class of abstract functions that will serve to modify the elements actually in the environment. They can be thought as correcting capabilities or as informal (textual) means to derive new elements from old ones. CREATE and MODIFY functions differ in that the USER elements are (more or less) complete specifications in the first and enumerations of changes in the second.

OPERATE is the class of abstract functions that will serve to realize activities such as simulation, in which some of the environment elements interact to produce new ones.

After reviewing the basic formalism proposed (the simulation language) we will give a more detailed account for the functions that MULTI must provide.

The model heading specifies the name given to the model, a set of Input and Output variables and, optionally, a time unit. Recursive nesting lies in the fact that component models share the same specification pattern. A convenient way to represent the structure of this kind of models is the given in figure 3 for an abstract example. In the figure, M2, M3, M4 are component models of M1, and they are coupled through variables V1, V2, V3, V4 of the global model (M1). At the same time, M5, M6 are component models of M2 and M7, M8 of M4. In this kind of structure (a tree) we will call "father" a node from which arrows leave, and the "sons" will be the nodes where these arrows arrive. In the figure, M1 is the father of M2, M3 and M4. Observe that a son can also be a father (i.e.: M2), and is this fact which characterizes recursive nesting. Nodes without sons represent elemental (not composed) models.

The fact that a model is composed of models is reflected in two ways:

a-The specification for each component model precedes the model equation, following the principle: "nothing can be used, which has not been defined".

b-The model equations include at least an equation of type M (model reference) for each component model.

The specification for M1 would be as follows:

```
MODEL M1
 MODEL M2 (INPUT I; OUTPUT O)
   Specification for M5,M6
   M2 equations
 END_MODEL M2
 MODEL M3 (INPUT I1,I2; OUTPUT O)
   M3 equations
 END_MODEL M3
 MODEL M4 (INPUT I; OUTPUT O1,O2)
   Specification for M7,M8
   M4 equations
 END_MODEL M4
EQUATIONS
 .......
 M  M2(V4,V1)
 M  M3(V1,V2,V3)
 M  M4(V3,V2,V4)
 ........
END_EQUATIONS
END_MODEL M1
```

Any programmer will quickly notice the similarities between the procedure or subroutine concept and the models as defined here. In fact, equations of type M are analogous to procedure calling conventions in programming languages like PASCAL. The variables specified as "parameters" of the model reference are attached to the Input-Output variables given in the model specification heading. The coupling of models, given by the horizontal arrows in figure 3, is accomplished through variables of the "father" model. In the example, M2 attaches its input variable I to V4, and V4 is also attached to the output variable O2 of M4. This allows that the coupling can be done through a set of equations of the father model.

The representation of model structure with a digraph like that of figure 3 serves three purposes:

1-Clarifies the hierarchic nature of model structure

2-Sketches the main interactions and localizes them at the proper level in the hierarchic structure.

3-Eases the discussion of locality concepts

Locality concerns the scope of a declaration or definition in a specification text. In our case, concerns the valid reference to variables or models inside the equations of a given model. We give, in axiomatic style, the rules of locality for H-DYN.

L1-A variable is defined when its value is determined, that is, when it appears at the left side of an equation (level, rate or auxiliar), or as an input in the model heading, or as attached to an output variable of a referred component model (in an equation of type M).

L2-A variable can not be referred outside the set of equations in wich it has been defined. Putting it in another way, if the same name appears in two sets of equations, belonging to different models, it identifies two distinct variables whose values are uncorrelated. For example, the name I appears both in M2 and M4 headings, identifying two uncorrelated variables, each one belonging to a model.

L3-A parameter can be defined either within the model equations or in the applicable experimental conditions. In either case, a parameter is only defined for the model equations and for the equations of the nested models (the "sons"). In the example, parameters defined in M1 can be used in M2 and even in M5 equations.

L4-A model can only be referred within its father equations. In our example, the model M5 can only be referred to inside the M2 equations.

L5-When a component model is referred more than once, it is assumed that a new copy is made for each reference.

The last two statements stablish a great difference between the programming concept of procedure and that of model proposed here. A component model can not be reused at different points by simply referencing it, it must be duplicated. The reason for this

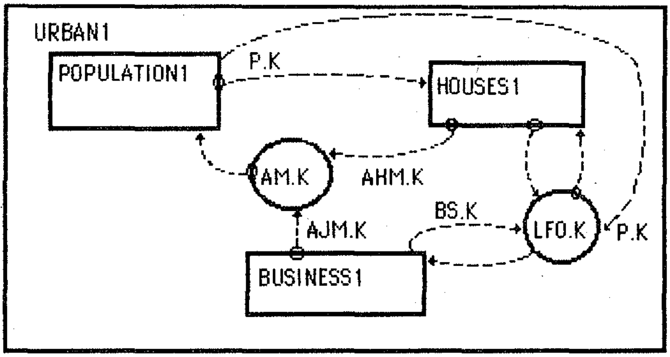

figure 4 Modular structure of URBAN1

lies in the following fact: When the model is being simulated, each copy of a model can be in a distinct state and all of them are concurrently evolving.

To end this section we give an example in figures 4 and 5. The example is the coding for the URBAN1 model, from Alfed & Graham (1976). Figure 4 shows the modular structure, in a flow diagram style, where the inner contents of the submodels have been hidden. Figure 5 shows the H-DYN code.

```
MODEL URBAN1 : TIME=YEARS
  MODEL HOUSES1(INPUT P.K,LFO.K; OUTPUT AHM.K,H.K)
  EQUATIONS
  L  H.K=H.J+(DT)(HC.JK-HD.JK)
  R  HC.KL=H.K*HCN*HCM.K
  A  HCM.K=HLM.K*HAM.K
  M  HOUSE_LAND (LFO.K,HLM.K)
  M  HOUSE_AVAL (HAM.K, HHR.K)
  A  HHR.K=P.K/(H.K*HS)
  M  ATRAC_HOUSES (HHR.K, AHM.K)
  END_EQUATIONS
  END_MODEL HOUSES1

  MODEL BUSINESS1(INPUT P.K, LFO.K; OUTPUT AJM.K, BS.K)
  EQUATIONS
  L  BS.K=BS.J+(DT)(BC.JK-BD.JK)
  R  BC.KL=BS.K*BCN*BCM.K
  A  BCM.K=BLM.K*BLFM.K
  M  S_BLM (LFO.K, BLM.K)
  M  S_BLFM (LFJR.K, BLFM.K)
  R  BD.KL=BS.K*BDN
  A  LFJR.K=LF.K/J.K
  A  LF.K=P.K*LPF
  A  J.K=BS.K*JBS
  M  ATRAC_JOBS (LFJR.K, AJM.K)
  END_EQUATIONS
  END_MODELS BUSINESS1

  MODEL POPULATION1 (INPUT AM.K; OUTPUT P.K)
  EQUATIONS
  L  P.K=P.J + (DT)(B.JK+IM.JK-D.JK-OM.JK)
  R  B.KL=P.K*BN
  R  D.KL=P.K*DN
  R  IM.K=P.K*IMN*AM.K
  R  OM.K=P.K*OMN
  A  PPG.K=(B.JK+IM.JK-D.JK-OM.JK)/P.K
  END_EQUATIONS
  END_MODEL POPULATION1
```

figure 5 H-DYN code for URBAN1 (Cont.)

```
EQUATIONS
M  HOUSES1 (POP.K, LAND_OCUP.K, ATRAC_HOUSES.K, HOUSES.K)
M  BUSINESS1(POP.K, LAND_OCUP.K, ATRAC_JOBS.K, BSNSS.K)
A  LAND_OCUP.K=(HOUSES.K*LPH + BSNSS.K*LBS)/AREA
A  ATRACTION.K=ATRAC_HOUSES.K*ATRAC_JOBS.K
M  POPULATION1 (ATRACTION.K, POP.K)
END_EQUATIONS
END_MODEL URBAN1
```

figure 5 H-DYN code for URBAN1

The following remarks are in order to the complete understanding of the example. The expression TIME=YEARS in URBAN1 heading means that the evolution of the system is on a years scale, so every time constant (specially DT) must be undestood as given in years units. Multipliers are represented as models, and their specification has been omissed in the figure. In URABAN1 the models HOUSE_LAND, HOUSE_AVAL, ATRAC_HOUSES, S_BLM, S_BLFM and ATRAC_JOBS are in fact multipliers. In the context of MULTI such incomplete specifications can be accepted, waiting for further specifications that will fill in the holes. Also, things like initial values for the state (level) variables or parameter values can wait for further specification or for the experimental conditions specification.

4-MODEL MANIPULATION AND DOCUMENTATION

In this section we will precise the set of functions (TOOLS) provided for the handling of the set of models in an environment. The functions described here are mainly like text manipulation functions, this is so because functions like automatic simplification/elaboration require a deeper research into the characteristics of System Dynamics models. Also, some of them are in fact families of functions, since we will not go into the more detailed enumeration of functions intended for MULTI. Figure 6 shows the functional digraph for the TOOLS (6a) and PREDICATES (6b) described here.

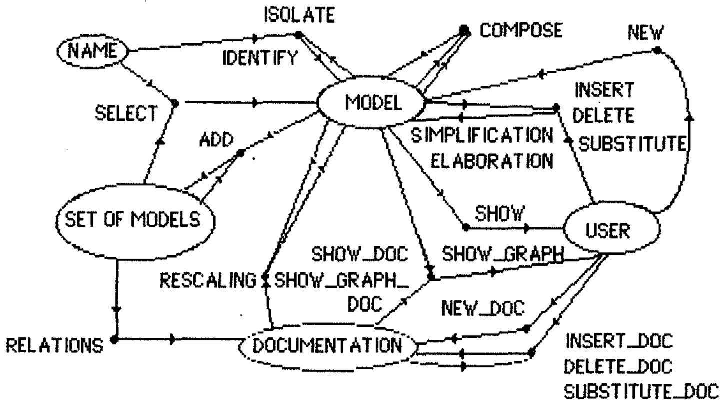

figure 6a TOOLS for model handling

Before going on into the discussion of function meanings, we must precise the meaning of the sets involved. "Name" is the set of all possible names (here model names) given for any environment actual state. "Set of Models" is the set of all possible sets of models for any environment. "Model" is the set of all possible individual models. "User" is a set of texts partitioned into several classes, each one fitted for a function. Finally, "Documentation" is a set of informations that can be about model elements or about relations between models. This last set deserves a more detailed description.

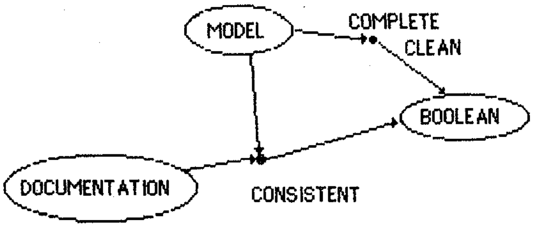

figure 6b PREDICATES for model handling

Model elements documentation concerns variables and parameters. For each element, we can record : its kind (variable, parameter), its type (level, rate, auxiliar), its functionality (input, output, state) and its dimension and units. Here, by dimension we mean a class of variables or magnitudes (sucha as time or length in physics). Associated with each dimension is a set of measure units (such as second, hour and so for the time dimension) in which a value of a measure in this dimension can be expressed. Also these units serve to interpret the values of a variable declared in that dimension. The insterest of all this is given by our ability to relate dimensions through dimensional equations (such as speed=space/time). This provides us with a mean to test the internal consistency of equations. Also, units can be related, and that gives us a second semantic information level to analize the equations. The definition of dimensions, units and their interrelation is given as documentation at the environment level, not for each model, and can be incomplete. Risking to look a little like pedants, we will illustrate the process of consistency evaluation. Suppose that the models in (Alfed & Graham 1976) constitute the set of models in an environment called URBAN_DYNAMICS. A feasible dimension declaration for it would be as in figure 7

```
DIMENSION DECLARATION
   DIMENSION FIRMS UNIT BUILDINGS
   DIMENSION LAND  UNIT ACRES
   DIMENSION PEOPLE  UNIT PERSONS
   DIMENSION WORK  UNIT JOBS
   DIMENSION UNEMPLOIMENT_PRESSION=PEOPLE/WORK  UNIT UP
   DIMENSION HOUSING  UNIT  HOUSEHOLDS
   DIMENSION  HOMES  UNIT FAMILIES
END_DIMENSIONS
```

figure 7 A dimension declaration for URBAN_DYNAMICS

After this declaration has been done, we can attach to each variable a dimensional meaning, and then prove the consistency of the equations under these declarations.

Suppose the following declarations:
LFJR : variable, auxiliar, UNEMPLOIMENT_PRESSION
(1) LF : variable, auxiliar, PEOPLE
J : variable, auxiliar, WORK
Under this declaration, in the model BUSINESS the equation
(2) LFJR.K=LF.K/J.K
is interpreted dimensionally as
(3) (LFJR) UNEMPLOIMENT_PRESSION= (LF) PEOPLE / (J) WORK =
=(LF/J) UNEMPLOIMENT_PRESSION
so the equation is dimensionally consistent.
suppose also that WORK has, besides JOBS, another measure unit defined as
(4) UNIT TEAM_JOBS=10*JOBS
that is, a TEAM_JOB is equivalent to ten JOBS. Suppose that in (1) J is now declared as:
(5) J : variable, auxiliar, WORK, TEAM_JOBS
Implicitly UP, the declared unit for UNEMPLOIMENT_PRESSION, is defined as
(6) UP=PERSONS/JOBS
With all this stated, the equation in (2) remains dimensionally consistent, but when looked in terms of units we find that:
(7) (LFJR) UP $\neq$ (LF) PERSONS / (J) TEAM_JOBS =
= (LF/J) PERSONS/TEAM_JOBS
We will call "rescaling" the reformulation of the equations to attain a version whose units are well arranged : a "scale consitent" formulation. In the example, we can make the following transformations on J:
(8) (J) TEAM_JOBS = (J*10/10) TEAM_JOBS = (J *10) TEAM_JOBS/10=
(J * 10) JOBS
so the scale consistent version of (7) is
(9) (LFJR) UP = (LF) PERSONS/(J*10) JOBS =(LF/(J*10)) PERSONS/JOBS
= (LF/(J*10)) UP
and in order to get an "scale consistent" model we must substitue (2) by
(10) LFJR.K=LF.K/(J.K*10)

The second kind of information we will record about models is relative to their relations. We notice three categories of relations between models:

a-Composition: a model is composed or is component of other models. This relation is given by the type M equations, in the specification texts of the models.

b-Transformation: a model is a transformation of another when it has been defined by modifying the latter (adding, deleting or changing variables and equations by application of modifying functions)

c-Simplification/elaboration: given by an homomorphic relation between models.

The simplification/elaboration relation between models can be approached from thre views:

1-Given two models, automatically prove that they are homomorphically related (one is a simplification of the other) and deduce the form of the relation (mappings between the state variables, parameters and functions of the two models)

2-Given a model and a class of homomorphisms, deduce the proper homomorphism to be applied to the model and get the homomorphic model. That is, given a model and a method of simplification or elaboration, get the simplified or elaborated model.

3-Over the set of models produced by the modeller activity, provide the means to document the incomplete mapping that the modeller has used intuitively to produce a model from a previous experience.

For us, the most feasible approach is the last one. It allows the modeller to specify informal and incomplete intuitions about the simplification/elaboration process. In fact, we think that the information gathered through these documentation tools, during diverse modeling processes, will be of precious value in the first steps of the search for the realization of the former approaches.

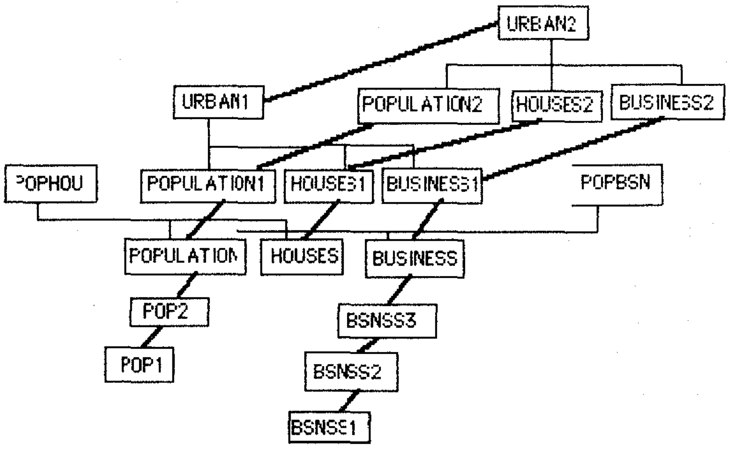

figure 8 URBAN_DYNAMICS relations between models

Composition and transformation relations can be easily stablished from the specification of models and the application of the provided transformation functions. Figure 8 shows pictorically these relations for the imaginary URBAN_DYNAMICS environment. The thin lines show composition relations (going down) and the thick oblique lines show transformation relations (going up).

Sometimes simplification relation lines go parallel to the lines for the transformation and composition relations. In figure 8, URBAN2 is someway analogous to the base model of the environment. The relation URBAN2 → URBAN1 is a simplification (aggregation), so we can attach to this relation some equivalences, such as the one between the population in URBAN1 and the sum of partial populations in URBAN2. Also, the relation URBAN2 → POPULATION2 is a simplification (isolation), but we can not say nothing analogous of BSNSS2 → BSNSS1. In fact, documentation about simplification/elaboration relations will be done in MULTI as relative to the transformation relations.

Now, we are in position to give meaning to the functions sketched in figure 6. We begin with the predicates of figure 6b.

COMPLETE : MODEL → BOOLEAN
CLEAN : MODEL → BOOLEAN
CONSISTENT : MODEL x DOCUMENTATION → BOOLEAN

A model will be complete (COMPLETE will give true when applied to it) if every variable has its value determined : there are not undefined variables. But there can be undefined parameters. A model will be clean when there are not ambiguities in any variable value, that is, there are not two ways to determine the value of a variable, for any variable of the model. A model will be consistent with the documentation if :

a-Every component model is consistent

b-For every documented variable there are no inconsistencies between the declaration and the use of the variable in the equations (no level variable used as a rate, and the like)

c-Every equation is dimensionally consistent

d-Every equation is scale consitent

<u>Display type functions</u>

SHOW : MODEL → USER
SHOW_DOC : MODEL x DOCUMENTATION → USER
SHOW_GRAPH : MODEL → USER
SHOW_GRAPH_DOC : DOCUMENTATION → USER

SHOW will provide a display for the model specification in its actual state of definition, including or not that of the component models. Notice that we have made abstraction of the actual device where the display will be done.
SHOW_DOC will provide the display for the model specification enriched with the unit and dimension declarations and the information relative to composition and transformation relations for the asked model.
SHOW_GRAPH will provide a graphical display for the model, a flow diagram or the curve specified by the points if it is a multiplier.
SHOW_GRAPH_DOC will provide graphs like the one in figure 8 or the one in figure 3.

<u>Create type functions</u>

NEW : USER → MODEL
NEW_DOC : USER → DOCUMENTATION

NEW will accept an specification text (can be incomplete) for a model, parse and translate it into an internal representation, integrating it into the environment.
NEW_DOC will give the way to create new pieces of documentation about a model, a relation between models or an environment characteristic (dimensions, units,...)

Modify type functions

INSERT : MODEL x USER → MODEL
DELETE : MODEL x USER → MODEL
SUBSTITUTE : MODEL x USER → MODEL
SIMPLIFICATION : MODEL x USER → MODEL
ELABORATION : MODEL x USER → MODEL
INSERT_DOC : DOCUMENTATION x USER → DOCUMENTATION
DELETE_DOC : DOCUMENTATION x USER → DOCUMENTATION
SUBSTITUTE_DOC : DOCUMENTATION x USER → DOCUMENTATION

INSERT introduces an incomplete specification (a set of equations) into a model specification, DELETE removes the equations and references for a given list of variables or parameters, SUBSTITUTE changes equations, references or components in a modelThese three functions are the trnasformation funtions mentioned before. The like apply to INSERT_DOC, DELETE_DOC and SUBSTITUTE_DOC relative to the documentation of models.
SIMPLIFICATION and ELABORATION are mentioned here in an attesting way. The preceding section has settled our approach to the problem of simplification/elaboration.

Operate type functions

COMPOSE : MODEL x MODEL → MODEL
IDENTIFY : MODEL x NAME → MODEL
ISOLATE : MODEL x NAME → MODEL
SELECT : SET_OF_MODELS x NAME → MODEL
ADD : SET_OF_MODELS x MODEL → SET_OF_MODELS
RESCALING : MODEL x DOCUMENTATION → MODEL
RELATIONS : SET_OF_MODELS → DOCUMENTATION

COMPOSE introduces a model into another as a component model.
IDENTIFY asigns a model specification to the name of a model that can be unspecified (this situation can arise from an incomplete specification). If the name identifies an already specified model, this asignement will suppose a kind of specification substitution.
ISOLATE is the inverse of the COMPOSE function, it extracts a component model from the gobal one where it is inmersed.
SELECT allows to select the model with which we will work from within the actual set of models in the environment
ADD acts conversely to SELECT, returning a model into the set of models
RESCALING performs the rescaling of a model not scale consistent
RELATIONS attest for the source of model relation documentation, in fact documentation will be updated automatically as create, modify and operate functions are performed.

5-REAL SYSTEM DATA AND DOCUMENTATION

Relative to the real system, MULTI will maintain two kinds of informations. Firstly the real system data, which is a collection of time trajectories recorded from the real system observation. These data are organized by the experimental conditions of their observation, so any access to them must be done through the proper experimental condition. Experimental conditions will be discussed later. Secondly, some documentation can be recorded which can be of interest in the modelling process, and as contrast for certain tests. This documentation will include for each system:

a-The documentation for all the subsystems or sectors that can be distinguished intuitively.

b-A list of "observables", "controllables" and "uncontrollables". These are the points at which we can observe the system behavior. "Observables" refer to the system outputs. "Controllables" refer to the inputs or system conditions that can be acted upon by the experiment operator. "Uncontrollables" are the unreachable inputs or system conditions, those that the operator can not control.

c-Information about the dimensionality and units for the observables, controllables and uncontrollables.

d-The enumeration of a set of aspects of the system, each of them supposes an approach to the system modelling. These aspects may or may not coincide with the sectors given in a.

e-For each aspect, the modeller can try a succession of models of increading refinement. So a set of models can be enumerated for each aspect.

Figure 9 shows the functions associated with real system data and documentation. There are no predicates associated. It deserves special mention the fact that every access to system data, for display, creation or modification, is made through the experimental conditions set.

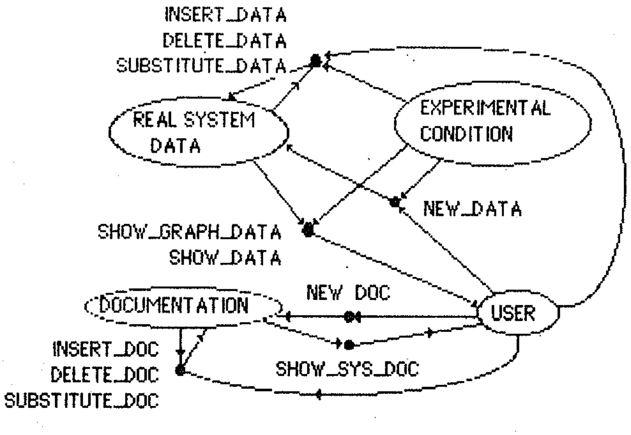

figure 9 Functions for real system manipulation

Display type functions

SHOW_GRAPH_DATA : REAL_SYS_DATA x EXP_COND → USER
SHOW_DATA : REAL_SYS_DATA x EXP_COND → USER
SHOW_SYS_DOC : DOCUMENTATION → USER

As in the case of model manipulation we have two main display modes : textual and graphical. The former is provided by SHOW_DATA, which displays the data observed under a experimental condition as sequences of numbers, and SHOW_SYS_DOC, which gives the textual display for the system documentation gathered until the displaying instant. The graphical mode is provided by SHOW_GRAPH_DATA which will give the graphical representation for the data observed under an experimental condition.

Create type functions

NEW_DATA : EXP_COND x USER → REAL_SYS_DATA
NEW_DOC : USER → DOCUMENTATION

The introduction of new observations through NEW_DATA into the real system data base will be done textually in principle, though other methods (graphical for example) can be thought to be provided. The new data is introduced under a certain experimental condition.

Modify type functions

INSERT_DATA : USER x EXP_COND x REAL_SYS_DATA → REAL_SYS_DATA
DELETE_DATA : USER x EXP_COND x REAL_SYS_DATA → REAL_SYS_DATA
SUBSTITUTE_DATA : USER x EXP_COND x REAL_SYS_DATA → REAL_SYS_DATA
INSERT_DOC : DOCUMENTATION x USER → DOCUMENTATION
DELETE_DOC : DOCUMENTATION x USER → DOCUMENTATION
SUBSTITUTE_DOC : DOCUMENTATION x USER → DOCUMENTATION

As for the create type functions, the modification of data will be done as a text modification of the sequences of numbers, but it will be desirable to provide more comfortable modes. Save for this observation, modify type functions are intuitively clear and not deserve more detailed description.

6-EXPERIMENTAL CONDITIONS

We transfer Zeigler's concept of experimental frame into the System Dynamics domain, and give it the name of "experimental conditions" with the aim of clearly detach the general approach of Zeigler, from the more specific attempt bounded to System Dynamics proposed by us.

Within the set of possible experimental conditions we will differentiate two classes real system and model oriented experimental conditions. This classification will not be made explicit at the function presentation, and its reason to be lies in the ability we have in each case to specify the experiment. For example, in the model experimentation we must specify initial values for the state variables, whether in the real system they are usually unreachable and unobservable. To be precise, experimental conditions for real system experimentation will be given by :

a-Time specification : lower and higher limits for the time interval observed and frecuency of observation.

b-A list of observables.

c-The values or time trajectories set for the controllable variables.

d-The values or time trajectories observed for the uncontrollable variables.

And the experimental conditions for model documentation will be given by:

a-Time specification : lower and higher limits for the time interval simulated. The frecuency of observation is specified by the model itself.

b-A list of variables to be recorded, in which non output declared variables can be included.

c-Values for parameters not given into model equations, or that must be modified for the simulation at hand.

d-Values or time trajectories for the input variables of the model. These can be real system observations.

e-Initial values for the state variables (levels)

f-The experimental conditions for the component models.

To illustrate these definitions figure 10 shows the experimental conditions for URBAN1. Notice that experimental conditions may have the same name as models. Experimental conditions need not to be complete in their specification of model or real system conditions.

```
EXPERIMENTAL CONDITIONS URBAN1

  START_TIME=0; STOP_TIME=150

  EXPERIMENTAL CONDITIONS  POPULATION1
  RECORD P.K,PPG.K
  PARAMETERS
    PN=50000; DN=0.015; BN=0.03; IMN=0.1; OMN=0.07
  INITIAL STATES
    P=PN
  END_CONDITIONS  POPULATION1

  EXPERIMENTAL CONDITIONS USINSS1
  RECORD  BS.K; BC.JK; BD.JK
  PARAMETERS
    BDN=0.025; LPF=0.35; JBS=18; BCN=0.07; BSN=1000
  INITIAL STATES
    BS=BSN
  END_CONDITIONS  BUSINESS1
```

figure 10 Experimental conditions for URBAN1(Cont.)

```
EXPERIMENTAL CONDITIONS HOUSES1
  RECORD H.K, HHR.K
  PARAMETERS
    HN=14000; HCN=0.07; LPH=0.1; MDN=0.015; HS=4
   INITIAL STATES
     H=HN
  END_CONDITIONS HOUSES1
RECORD LFO.K
PARAMETERS
  AREA=10000; LPH=0.1; LBS=0.2
END_CONDITIONS URBAN1
```

figure 10 Experimental conditions for URBAN1

Figure 11 shows the functional diagram for experimental conditions. The reader may note the similarities betwen figures 6a and 11a. These similarities are due to the existence in the set of experimental conditions of composition and transformation relations similar to the ones discussed for the models. In fact, if we consider a set of experimental conditions, with only one of them for each model, in URBAN_DYNAMICS environment, we could depict a graph quite similar to that of figure 8, but for experimental conditions instead of models. As with the models we begin with the predicates description.

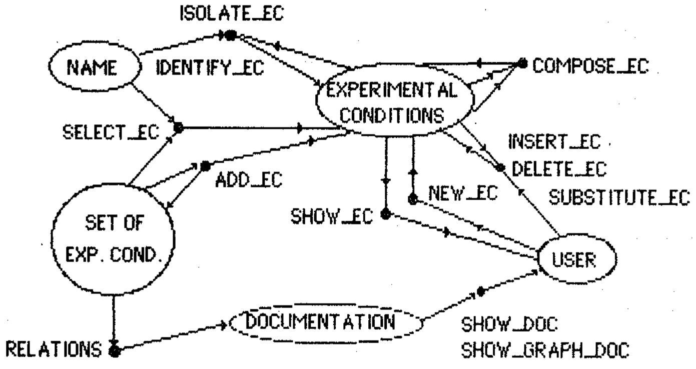

figure 11a TOOLS for experimental conditins manipulation

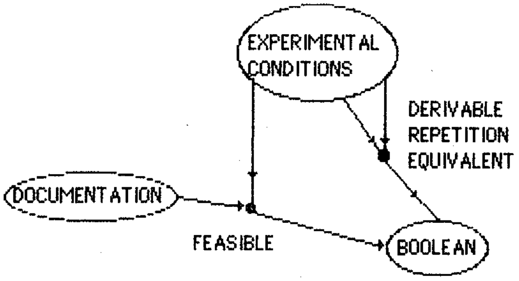

figure 11b PREDICATES for experimental conditions manipulation

DERIVABLE : EXP_COND x EXP_COND → BOOLEAN
REPETITION : EXP_COND x EXP_COND → BOOLEAN
FEASIBLE : EXP_COND x DOCUMENTATION → BOOLEAN
COMPLETE : EXP_COND x DOCUMENTATION → BOOLEAN
EQUIVALENT : EXP_COND x EXP_COND → BOOLEAN

The original meaning for derivability of experimental frame is that a frame E' is derivable from a frame E if the data gathered in frame E' can be deduced from the data gathered in frame E. Put it in another way, frame E' is more restrictive than frame E or there is some kind of mapping between the elements of both frames. When transferred to our present context, we attach the following meaning to DERIVABLE : E' is derivable from E if any or some of the conditions that follow apply:

D1-The time interval in E' is a proper subinterval of the specified in E

D2-Everything equal (save, maybe, D1) the list of observables or variables to be recorded in E' is a sublist of that in E.

D3-Everything equal (save, maybe, D1 or/and D2) the set of parameters or controllable variables determined in E' includes that of E (Note the change in the inclusion direction)

D4-Everything equal (save, maybe, D1,D2 or/and D3) the set of inputs or uncontrollable variables determined in E' includes that of E.

D5-Everything equal (save, maybe, D1, D2,D3,D4) the set of state variables with initial values in E' includes that of E.

D6-Everything equal (save, maybe, D1,D2,D3,D4,D5) at least one of the experimental conditions for the component models or sybsystems in E' is derivable from that in E.

DERIVABLE stablishes a partial ordering on the set of experimental conditions. This ordering is relative to the amount of information that can be gathered through each experimental condition. Under this interpretation, E' is derivable from E if its associated data set is a subset of that of E. And that gives meaning to the inclusions mentioned in D1 to D5.

REPETITION stablishes an equivalence relation between experimental conditions. The classes must be understood as experiments factorization. More precisely, two experimental conditions are repetition one of the other if, everything equal, they differ only in that :

R1-The lower and higher limits for the time interval of observation are shifted by a constant (the interval length remains the same) and/or

R2-A subset of the parameters or controllable variables have changed their values or time series , and/or

R3-A subset of the inputs or uncontrollable variables have changed their values or time series, and/or

R4-A subset of the state variables have changed their initial values, and/or

R5-We can find at least that one of the subsystems or component models is a repetition.

The classes given by repetition have at least one element, since every experimental condition is a repetition of itself. Usually, these classes will be sets whose elements specify a series of experiments intended to anlise the system behavior causes or the sensitivity of the system. We can attach to each class a representative element, this will be the experimental conditions obtained by suppression of the variables and parameters whose values are changing within the class elements and/or specifying a time interval which includes all the shifted intervals. This representative experimental condition will have the property that every experimental condition of the class is derivable from it and, besides that, the union of the data sets for all the class elements will be a subset of the representative element data set. We will not go further, but the reader could notice that this mechanism gives way to a hierarchical method for the design of experiments.

An experimental condition is FEASIBLE at the present state of the environment, given the documentation we have recorded, if all its elements are defined. This means not that the conditions can be applied to any model or system, but only that it don't introduces anything properly new.

Two experimental conditions are EQUIVALENT, given that one is a real system condition and the other is model condition, if all their elements coincide or can be mapped one to one. This mapping must preserve characteristics such as dimensions and units. This means that their data sets are comparable.

The discussion of model manipulation serves as a good background in wich the reader may understand intuitively the meaning of the TOOLS provided for experimental conditions, so we will shorten their description.

<u>Display type functions</u>

SHOW_EC : EXP_COND → USER
SHOW_DOC : DOCUMENTATION → USER
SHOW_GRAPH_DOC : DOCUMENTATION → USER

Notice that the only documentation generated for the experimental conditions is that of the relations given by composition, transformation functions and by the predicates DERIVABLE, REPETITION and EQUIVALENT.

<u>Create type functions</u>
NEW_EC : USER → EXP_COND

<u>Modify type functions</u>
INSERT_EC : EXP_COND x USER → EXP_COND
DELETE_EC : EXP_COND x USER → EXP_COND
SUBSTITUTE_EC : EXP_COND x USER → EXP_COND

<u>Operate type functions</u>
COMPOSE_EC : EXP_COND x EXP_COND → EXP_COND
ISOLATE_EC : EXP_COND x NAME → EXP_COND
IDENTIFY_EC : EXP_COND x NAME → EXP_COND
SELECT_EC : SET_OF_EXP_COND x NAME → EXP_COND
ADD_EC : SET_OF_EXP_COND x EXP_COND → SET_OF_EXP_COND
RELATIONS_EC : SET_OF_EXP_COND → DOCUMENTATION

7 - SIMULATION, VALIDATION AND ANALYSIS

A central issue for multifacetted modelling was the formalization of simulation and validation concepts. In our design of MULTI, providing a unified access and support for simulation, validation and analysis of models is a major goal. In this section wee discuss a set of tools (of the operate type) and predicates which give a first approach to this goal. We begin with the functions related to simulation. Figure 12 shows the tools and predicates involved in simulation.

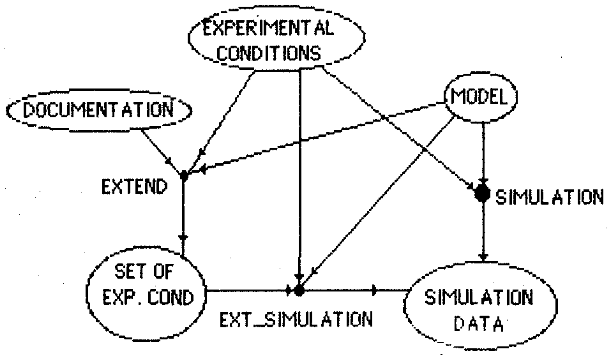

figure 12a TOOLS for simulation

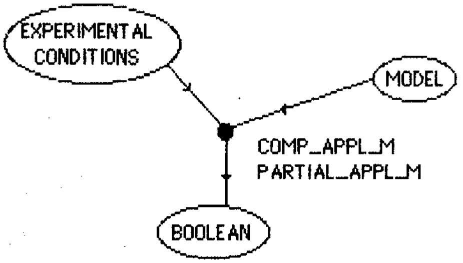

figure 12b PREDICATES for simulation

Predicates in figure 12b plus those in figure 6b, examine the conditions needed for a simulation realization. Thus, before trying to simulate a model under an experimental condition we must guarantee that the model is COMPLETE, CLEAN and CONSISTENT, and also that the experimental condition is completely or partially applicable to the model. Before going on we enumerate the functions and predicates involved.

COMP_APPL_M : MODEL x EXP_COND → BOOLEAN
PARTIAL_APPL_M : MODEL x EXP_COND → BOOLEAN
SIMULATION : MODEL x EXP_COND → SIMUL_DATA
EXT_SIMULATION:MODELxEXP_CONDxSET_OF_EXP_COND → SIMUL_DATA
EXTEND : MODEL x EXP_COND x DOCUMENTATION → SET_OF_EXP_COND

Complete applicability of a condition to a model, tested through COMP_APPL_M, means that the experimental condition sets the values for all the parameters, inputs and states so its application to the model produces a simulable object (without ambiguities or undefined terms). Notice that complete applicability depends as much of the model as of the experimental condition. The latter can be completely applicable to a model and not to another.

Partial applicability (PARTIAL_APPL_M) only guarantees that all the variables and parameters specified by the experimental condition appear within the model specification, but not that all the undefined terms (i.e.: inputs and parameters) have been determined. That can be undestood in two ways:

a-The experimental condition is designed for a class of models to which the actual model belongs.

b-The experimental condition is the representative element of a repetition class, whose elements are completely applicable to the actual model.

Assumed the second interpretation, we can realize that the simulation of the model under the partially applicable experimental condition is equivalent to the collection of the simulations made under all the repetitions of its class. We think of EXTEND as a funcion providing all the members of the repetition class and EXT_SIMULATION as the collection of simulation realizations for each member in the class. The reader will notice that EXTEND can be an extremely comnplex function, but we eill not pursue this further here.

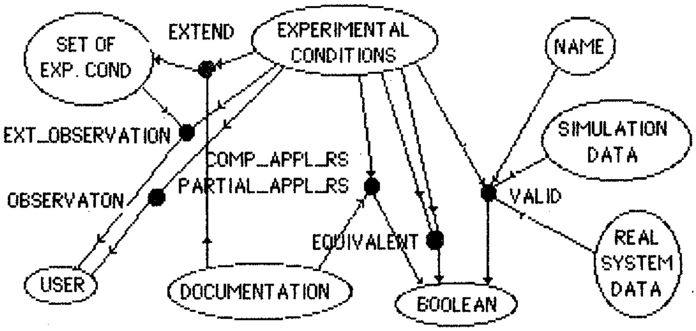

figure 13 Predicates and functions for validity

Figure 13 shows the predicates and functions involved in the validity issue. Observe that they are mainly predicates, and validity itself is a condition tested by the predicate VALID. We enumerate the functional formulation:

VALID : EXP_COND x NAME x SIMUL_DATA x REAL_SYS_DATA → BOOLEAN
COMP_APPL_RS : DOCUMENTATION x EXP_COND → BOOLEAN
PARTIAL_APPL_RS : DOCUMENTATION x EXP_COND → BOOLEAN
EQUIVALENT : EXP_COND x EXP_COND → BOOLEAN
OBSERVATION : EXP_COND → USER
EXT_OBSERVATION : EXP_COND x SET_OF_EXP_COND → USER

We propose two predicates (COMP_APPL_RS, PARTIAL_APPL_RS) to test the complete and partial applicability of experimental conditions to real systems. As for simulation, complete applicability is a condition for OBSERVATION, and partial applicability for extended observation (EXT_OBSERVATION), both concepts are analogous to the discussed before. The observation functions must not be interpreted as display functions, although their functional pattern induces to think so, they try to represent an activity which is properly done outside MULTI and their results are sequences of figures that will be introduced in the real system data set through the function NEW_DATA.

Validity is defined as a comparison between two data sets : the one obtained by simulation of a model under an experimental condition and the obtained by observation of the real system under an experimental condition. A previous condition for this comparison is that of real system and model experimental conditions equivalence. Given this we can discuss the issue of validity under an experimental condition (that of the real system). The kind of comparison between data sets can range from pure equality to the evaluation of some measure of "closeness". Different comparison methods will give way to different validity predicates. At the moment we have not still precise the possible methods intended for MULTI. An special case is that given by the validation of a repetition class data set. In this case, comparison could not be done in a one to one fashion, and some kind of averaging will be needed.

The last point in MULTI design, of which we have still not developed a functional description, is that of model analysis tools. This tools can range from the classical sensitivity analysis, to more recent developments such as the described by Aracil (1983) and Aracil & Toro (1984). The attraction of introducing these tools into MULTI will lie in the ease of use that would come from that, as well as in the growing capacity of the package.

8-CONCLUSIONS

A functional description for a software system (MULTI) is provided. This system is an attempt to bring Multifacetted Modelling ideas into the System Dynamics methodology. On going work tries to refine this description and to build a prototype system. Further research will be addressed to the integration within MULTI of methods for the qualitative analysis of models and other software tools, that can be of application to System Dynamics modelling.

REFERENCES

Alfed L.E., A.K. Graham (1976)
Introduction to Urban Dynamics
MIT Press

Aracil J. (1983)
Introducción a la Dinámica de Sistemas
Alianza Editorial

Aracil J., M. Toro (1984)
A case study of qualitative change in System Dynamics
Int. J. System Sci. Vol15 No 6 pp.575-599

Liskov B.H., S.N. Zilles (1975)
Specification techniques for Data Abstactions
IEEE Trans. on Soft. Engi. Vol SE-1 No 1 pp.7-19

Oren T.I., Zeigler B.P. (1979)
Concepts for advanced simulation methodologies
Simulation Vol 32 No 3 pp.69-82

Torrealdea F.J., M. Graña, F.Ferreres, J.J. Dolado (1986)
Hierarchical modelling in System Dynamics
EURO VIII Eighth European Conf. on Oper. Res., Lisbon, Portugal

Zeigler B.P. (1976)
Theory of Modelling and Simulation
John Wiley

Zeigler B.P. (1978)
Structuring the organization of partial models
Int. J. General Systems Vol 4 pp.81-88

Zeigler B.P. (1984a)
Multifacetted Modelling and Discrete Event Simulation
Academic Press

Zeigler B.P. (1984b)
Theory of Discrete Event specified Models : Modularity, Hierarchy, Experimental Frames
Int. J. General Systems Vol 10 pp.57-84