Guittet L., Girard P., & Pierra G. Dialogue verification using the EXPRESS language. *Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'97),* Eds. M.D. Harrison & J.C. Torres, *Granada, Spain,* vol. Conference proceedings,4-6 June 1997, pp. 457-471.

Dialogue verification using the EXPRESS language

L. Guittet, P. Girard, G. Pierra

Laboratoire d'Informatique Scientifique et Industrielle
Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Site du Futuroscope - B.P. 109 - 86960 FUTUROSCOPE Cedex - France
e-mail : {girard, guittet, pierra}@ensma.univ-poitiers.fr

Abstract

This paper presents some preliminary results in using the formal language EXPRESS to verify dialogues of Interactive systems. These results demonstrate that many static verifications can be achievedand that tools integrating EXPRESS verification should be very useful for a priori interactive system validation

Keywords

Computer-Human Interaction (CHI), Formal Methods, Computer Aided Design (CAD).

INTRODUCTION

Formal specification of interactive systems usually consists in defining a model associated with a formal language (Palanque 1992; Duke and Harrison 1993; Paterno' 1994). Our approach is slightly different. We have started from an existing model of interactive systems (the H^4 model) with existing tools for interactive systems building. The dialogue description in the model is made by extended ATN. The whole system has been developped for Computer-Aided Design (CAD) systems and is really suitable for them. Despite attempting reasoning upon these transition networks, we decided to use another formal language, in fact the EXPRESS language, and its associated tools, to verify some properties of the dialogue.

In the first two sections, we give a short description of the CAD domain and its specificities, and we explain the main characteristics of the H^4 approach. In the third section, we describe the main parts of the EXPRESS language, giving examples which are applied to our case. In the fourth section, we show how EXPRESS modelling allows us to verify many concrete propoerties. Last, we explain the limitations of the method and the development we planned for this work.

1. CAD systems specificity

CAD systems are powerful design systems that allow users to build complex models, such as mechanical models, architectural models, and so on. We will try, in this section, to expose the basis of CAD specificities that lent us to our specific model. We begin with tasks characteristics, and follow by object description. Our examples belong to the technical domain, but are really very simplified to avoid for readers'lack of background in this domain. We will describe our problem, related to the *Taxonomy for Interactive Graphics Systems* (Pierra 1995) resulting from discussions that took place during the Eurographics Workshop on Design, Specification and Verification of Interactive Systems’95 (Bonas, France), in the Working Group on Taxonomy

The major characteristic of technical or architectural design is its preciseness. Objects are completely and explicitely defined by users (lines are defined by their two end points, sweeps by their sweeped face and sweeping vectors, and so on). These explicit relations require users to express them. While relations should apply between distinct objects, users'tasks have to be **multi-object tasks**. More, because CAD systems are powerful, they allow users to perform structured tasks. These two points are illustrated on the following example:

Assume the goal of a user is: "I want to build a circle whose centre is the end of *this* line, and whose radius is the distance between *that* point and the end of *that* line". The expected result is shown on figure 1:

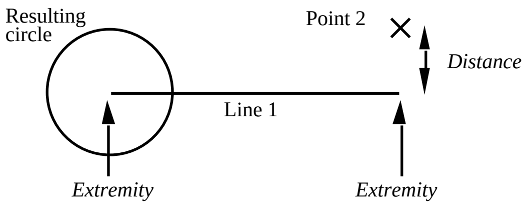

Figure 1: The user's goal

Applying Norman's theory of activity (Norman 1986) is straightforward. The goal/subgoal hierarchy which is expressed above may be mapped to the task/substask hierarchy shown on figure 2:

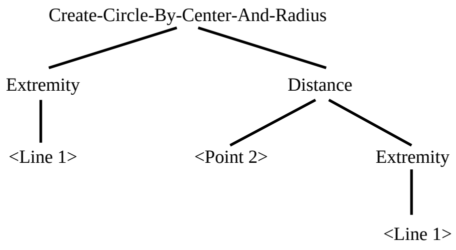

Figure 2: The corresponding task/subtask hierarchy

A classic CAD dialogue might be as shown in figure 3, which reflects this task/substask hierarchy. Commands are written in bold, and selected objects appeared between <>:

```
create_circle_centre_radius
  extremity
       <line_1>
  distance
       <point_2>
       extremity
            <line_1>
```

Figure 3: A structured task

This example allows us to classify the tasks: the *distance* subtask is intended to elaborate 'something' (that we call **token**) which is used as a parameter for the higher degree subtask (here the *distance*), which elaborates itself another token for another task. Producing any token which is not used as parameter by any task should be considered at least as an error of the user, and perhaps as a design error of dialogue. At the opposite, the *circle creation* task does not produce anything to be used by any higher-level task.

This particular point led us splitting tasks in two main categories: **production subtasks**: which ***produce*** some token for another task, and **terminal tasks**, which do not produce anything for task usage. For example, object management (creating, deleting, structuring, ...) or window management (zooming, translating, ...) are the realm of terminal tasks, while expressions (extremity, distance, numerical expressions, ...) are typical production subtasks. By extension, simple value inputs or object selection must be considered as production subtasks.

Conceptual objects are highly structured in CAD area. Solids are the inner part of closed shells; shells themselves consist of faces, and so on. When selecting a line, does the user want to select either that line, either the face it belongs, or the solid it limits? Depending on the dialogue process state, the answer may vary.

Conceptual objects are also relational objects. Hidden surface visualisation is a good example of this point: each object's visibility cannot be computed on its own. For all this reasons, it does not exist any direct mapping between conceptual objects and presentation objects.

2. The hierarchical interactors solution

According to the taxonomy, these systems support **multi-object** and **structured** tasks, and the conceptual objects are **structured** and **relational** objects. It is well known, in Software Engineering, that these complex systems, even if designed according to object oriented methods, have to be splitted up into subsystems. The Arch (or Slinky (Bass, Faneuf et al. 1992)) model precisely provides for such a macrostructuring of systems. But Software Engineering principles also require the interfaces, specification and relationships between these subsystems to be precisely defined, and require each system to correspond to one unique abstraction. According to this principle, we will describe our model by use of the Arch model.

We identified four components with hierarchical decomposition. Two components from the four exactly match with the components of the ARCH model: the functional core, and the dialogue component. The first one is in charge of conceptual objects hierarchy. The second one exactly reflects the task/subtask hierarchy. The presentation component is devoted to the pipeline viewing (both for inputs and outputs), and reflects the hierarchy of space transformations (PHIGS 1989; Carson 1993). Last, the interaction toolkit component implements the logical input devices whose hierarchic composition is generally accepted (Duce, Van Liere et al. 1990; Duke and Harrison 1993; Paterno' 1994). These four hierarchies explain the name we chose for our model.

The fifth ARCH component has to be detailed. It is the ***domain adaptor component*** that implements the exchange mechanism between the Dialogue Component and the Domain Specific Component. We propose using the term of **questionnaire** for modelling such an exchange mechanism.

For example, assume we want to design the syntactic call of the questionnaire for the "select entity" subtask. The semantic role of this subtask is to return the nearest entity from a given position (simple mouse click or elaborated 3D position). It may fall down if the model is empty. So, the questionnaire may be as follows, in an Ada like formalism:

```
Selected_Entity ?
Position:      IN     WC_Coordinate
Entity    :    OUT    Graphical_Entity
Status    :    OUT    BOOLEAN
```

Figure 4: A questionnaire for a production subtask

The domain adaptor component also insures the feedback of the corresponding procedures through a direct projection on the rendering functions of the presentation component. This direct access from the Domain Adaptator Component results from two points: (1) the decoupling of domain obects and tasks, and (2) the need for projecting the domain objects (alone or not) on the presentation area.

The following figure summarize the H^4 architecture model for interactive graphic application that support multi-object structured tasks and whose the objects are both structured and relational objects. The formal definition of this model may be found in (Guittet 1995).

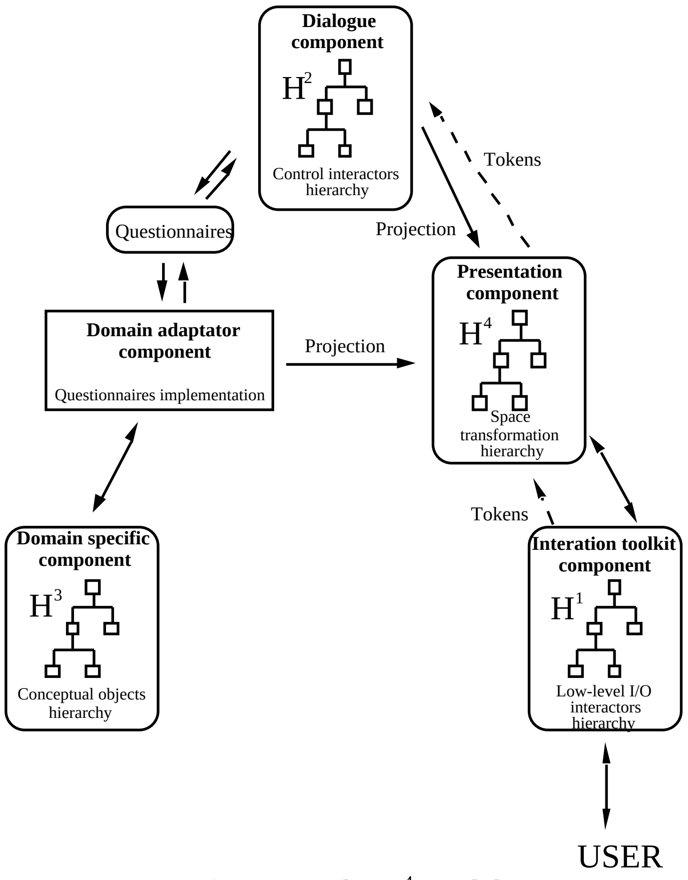
Dialogue component
H2
Control interactors hierarchy
Tokens
Projection
Questionnaires
Domain adaptator component
Questionnaires implementation
Projection
Presentation component
H4
Space transformation hierarchy
Tokens
Domain specific component
H3
Conceptual objects hierarchy
Interation toolkit component
H1
Low-level I/O interactors hierarchy
USER

Figure 5: The H^4 model

The H^4 architecture model has been used to develop a complete Computer Aided Design system for mechanical purpose that uses the Motif toolkit. The dialogue interactors are specified in an ATN-based specification language and the complete dialogue component is generated as an Ada program.

The application structure can be synthesized as in figure 6, which focusses on the dialogue component: we can see the hierarchy of interactors, and the monitor that manages tokens. This structure will be modelized in the EXPRESS language in the next sections.

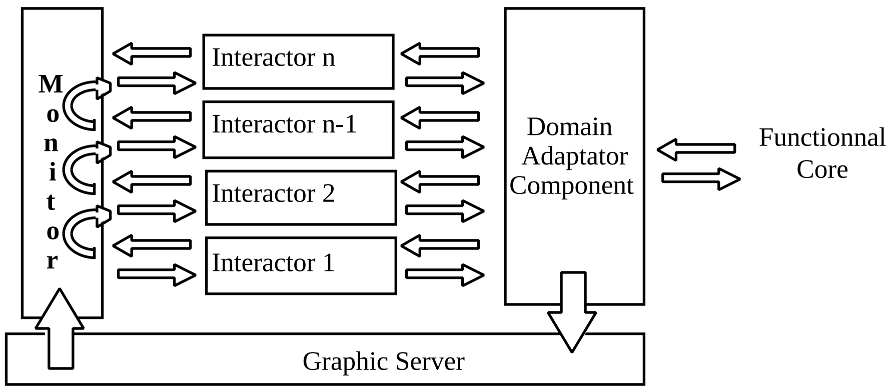

Figure 6: The hierarchical interactors

3. The EXPRESS Language

The EXPRESS language has been developed to describe the information required to design, build, and maintain products (Schenk and Wilson 1994). The langage, described in

(EXPRESS 1994), is since September 1994 an international standard. In this section, after having given a short overview of the EXPRESS language, we explain some of its feature using examples of our system.

3.1. Overview

This modelling language, used in STEP (Standard for the Exchange of Product Model Data), is object-based (as opposed to Entity-Relationship modelling language) and syntactically compilable but yet, non-executable. EXPRESS has been developed in parallel with its usage, which gave it a pragmatic flavour. This language only deals with the specification of the static aspects of data.

The major concepts used to describe models are entity types, supertypes or subtypes, and constraints.

Entity types describe collections of **entity instances** with similar properties, common relationships and semantics. Entity instances belonging to the same entity type have the same attributes. Each entity instance is uniquely identifiable by definition (two entity instances with the same attribute values can be distinguished).

Attributes describe the properties of entity types. Attribute values are either values of simple or defined types, or instances of entity types (a "has-a" relationship). The three kinds of attributes are:

- **explicit**: this property has a static value, which is independent from other attribute values.
- **derived**: the value is a result of a function over other attribute values.
- **inverse**: the value represents the coupling between "subject" entiies and "user" entities that refer the subject (Schenk and Wilson 1994).

Supertypes define a "is-a" relationship between entity types and one or more refined versions of them. Refined entities are called **supertypes** and every refined versions are called **subtypes**. Subtypes inherit attributes and constraints from their supertype (in direct and transitively indirect mode), and also add specific attributes or restrict supertype properties. **Inheritance** refers to the mechanism of sharing properties and constraints using the supertype structure. The supertype relationship is transitive.

Constraints, used to restrict entity instances or values, are **local rules** (applied to every individual instance) or **global rules** (applied to entity instances of different entity types or to particular entity types). Unlike most data modeling formalisms that only capture cardinality or set-oriented constraints on data conforming to data models, EXPRESS enables modeling any kind of constraints. Thanks to several built-in functions, and to a pascal-like procedural language, **functions** may be defined. These functions, in turn, may be used to define constraints. **Uniqueness** constraints restrict attributes or set of attributes to have unique values within the extent of entity types. Inverse attributes constrain the existence dependencies between entity instances of different entity types.

3.2. Simplified example

3.2.1. The token tree

The following example shows the main features that can be found in an EXPRESS specification.

```
ENTITY token  ; ABSTRACT
  SUPERTYPE OF (ONE OF(command, parameter));
END_ENTITY ;

ENTITY command
  SUBTYPE  OF (token) ;
       name : STRING ;
  UNIQUE name;
END_ENTITY ;

ENTITY parameter;  ABSTRACT
  SUBTYPE  OF (token) ;
  SUPERTYPE OF (
     ONE OF(number, position,object)) ;
END_ENTITY ;
```

```
ENTITY number
  SUBTYPE  OF (parameter) ;
       value : REAL ;
END_ENTITY ;

ENTITY position
  SUBTYPE  OF (parameter) ;
       point : Coordinate ;
END_ENTITY ;

ENTITY object;
  SUBTYPE  OF (parameter) ;
  SUPERTYPE OF (ONE OF(line, circle,...)) ;
  reference : BINARY ; -- in the data base
END_ENTITY ;
```

Figure 7 : The EXPRESS specification of the token hierarchy

In this EXPRESS specification, the ENTITY token is the root of an inheritance tree (Fig 8).

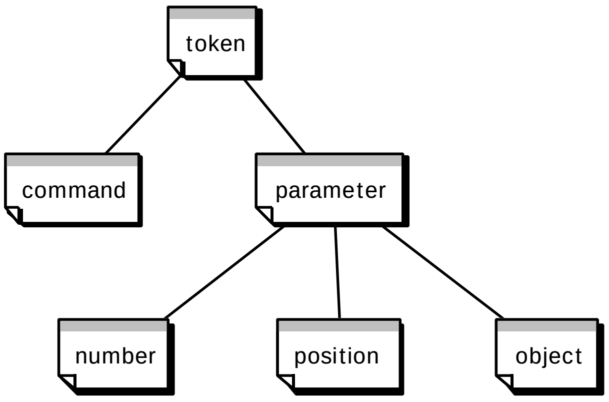

Figure 8 : The token hierarchy

This tree, which results form the SUPERTYPE and SUBTYPE clauses, describes the lexical elements ot the language. Commands have a name which is an attribute. They should be considered as the keywords of the language. Their unicity is assumed by the UNIQUE clause. The parameters, which are specialized in numbers, positions or system objects (here, lines and circles), have each a specific attribute: their associated value.

3.2.2. System description

```
ENTITY system;
        tokens: SET [0:?] OF token ;
        base_production: SET [0:?] OF token ;
        hierarchy: LIST [0:?] OF interactor ;
WHERE
   base: base_production IN tokens ;
   production:QUERY ( i <* hierarchy / i.token_in IN
 base_production+production_behind(SELF,i)) = hierarchy ;
END_ENTITY ;
```

```
FUNCTION production_behind(
    s: system ;i : interactor ):SET OF token ;
        LOCAL tokens: SET OF token := [] ;
               ni :INTEGER := 1; END_LOCAL;
REPEAT UNTIL (i=s.hierarchy[ni]) ;
        tokens := tokens+ s.hierarchy[ni].product;
        ni:=ni+1;
END_REPEAT;
RETURN tokens ;
END_FUNCTION;
```

Figure 9 : The system entity

The system entity is composed of a set of used tokens (**SET**[0:?] **OF** token), a subset of producible tokens by basic I/O, and an ordered list of interactors (**LIST**[0:?] **OF** interactor). This order is defined by the production/consummation logic of tokens. The strict respect of the so-called ‘production’ rule (WHERE) ensures every interactor (QUERY()=hierarchy) to be able to receive tokens produced by lower interactors or by basic I/O. This rule uses the result of a function (FUNCTION production_behind) which calculates the whole set of produceable tokens from the interactors which are lower than the one which is given as parameter.

3.2.3. Interactors

```
ENTITY interactor ;
        name : STRING ;
        states : SET [0:?] OF state ;
        transitions : SET [0:?] OF transition ;
        initial_state : state ;
DERIVE
   token_in : QUERY ( t <* syst.tokens /SIZEOF ( QUERY (
             tr <* transitions / tr.token=t) #0) ;
  product : QUERY ( t <* syst.tokens /SIZEOF ( QUERY (
             tr <* transitions / tr.action.production=t) #0) ;
INVERSE syst : LIST [1:1] OF system for hierarchy ;
UNIQUE name ;
WHERE initial_state IN states;
END_ENTITY ;
```

Figure 10 : The interactor, system component

Every interactor is composed of his name and the states and transitions of its associated ATN. Two main characteristics are calculated by derivation (DERIVE) : the consumed tokens (token_in) and the produced parameters (product).

3.2.4. States, transitions and questionnaires

```
ENTITY state ;
        name : STRING ;
        prompt : OPTIONAL STRING ;
INVERSE inter : LIST [1:1] OF interactor for states ;
UNIQUE inter ,name ;
END_ENTITY ;

ENTITY questionnaire ;
        in_list : LIST [0:?] OF parameter ;
        production : OPTIONAL parameter ;
END_ENTITY ;

ENTITY transition ;
        s_begin, s_end : state ;
        key : token ;
        action : OPTIONAL questionnaire ;
INVERSE inter : LIST [1:1] OF interactor for transitions ;
WHERE
   inter_coherency :inter=begin.inter AND inter=end.inter ;
END_ENTITY ;
```

Figure 11: States, transitions and questionnaires

States are the stable states of interactors. Optional prompts are sent to users before inputs. Transitions, which connect initial states (s_begin) to terminal states (s_end), and which are associated with optional actions (OPTIONAL), are fired when the current token is the key attrobute. When the action is present, it is triggered by the functional core. Input parameters which have not been concumed are given to the action. Coherency verification between given and expected parameters will be detailed later.

3.3. System instances

The EXPRESS language is a data-oriented modelling language. So, it allows defining models. The above description is a partial view of the Hierarchical Interactors model.

The power of the EXPRESS environment is to provide for verifications against EXPRESS models. What is to be verified is called "instances" or "physical files". Next figure presents such a physical file, which supports the definition of the dialogue conforming to the task/subtask hierarchy of section 2.

Guittet L., Girard P., & Pierra G. Dialogue verification using the EXPRESS language. ***Eurographics Workshop on Design, Specification and Verification of Interactive Systems (DSV-IS'97),*** **Eds. M.D. Harrison & J.C. Torres,** ***Granada, Spain,*** **vol. Conference proceedings,4-6 June 1997, pp. 457-471.**

```
-- the parameters
#1=number()
#2=position()
#3=line()
#4=circle()
-- the questionnaires
#5=questionnaire((#3),#2)        --extremity of a line
#6=questionnaire((#2,#2),#1)     --distance p1,p2
#7=questionnaire((#3,#1),$)      --circle construct
-- the commands
#11=command('extremity')
#12=command('distance')
#13=command('circle_center_radius')
--the information interactor
#100=interactor('inform',(#101,#102),(#110,#111),#101)
#101=state('initial',$)
#102=state('extremity_of',$)
#110=transition(#101,#102,#11,$)
#111=transition(#102,#101,#3,#5)
```

```
--the calculator interactor
#200=interactor('calcul',
    (#201,#202,#203),(#210,#211,#212),#201)
#201=state('initial',$)
#202=state('distance_of',$)
#203=state('distance_of_p_and',$)
#210=transition(#201,#202,#12,$)
#211=transition(#202,#203,#2,$)
#212=transition(#203,#200,#2,#6)
--the creation interactor
#300=interactor('calcul',
    (#301,#302,#303),(#310,#311,#312),#301)
#301=state('initial',$)
#302=state('circle_by',$)
#303=state('circle_by_center_and',$)
#310=transition(#301,#302,#13,$)
#311=transition(#302,#301,#3,$)
#312=transition(#303,#300,#2,#7)
-- and the system
#1000=system(
  (#1,#2,#3,#11,#12,#13),(#2,#3),(#101,#102,#103))
```

Figure 12 : Physical file

The following table establishes the correspondance between actions an numbers in the physical file (#301 for example)

création	action								7
	state	301	302		303				301
calcul	product								1
	action								6
	state	201				202	203		201
inform	product				2				2
	action				5				5
	state	101		102	101			102	101
user sentence			13	11	3	12	2	11	3

4. Verifications and Validation

Our goal was to use the EXPRESS language to verify the correctness of dialogues. Checking physical files against well-written EXPRESS models ensures properties. In this section, we explain what kinds of properties we have checked with our models, and then, we present the first results that validate our approach.

4.1. Verifications

Our EXPRESS model of hierarchical interactors verifies two kinds of properties: individual properties on each interactor, and global properties on the whole hierarchy, over token exchanges.

4.1.1. Individual properties

Different levels of verifications can be achieved. Some of them have been yet detailed in the previous section:

- at structural level, it is easy to ensure the connectivity and the state reachability in the automata. The "WHERE" rules ensure such verifications,
- at lexical level, we can verify that every command and parameter which are defined in its input list is actually used in each interactor,
- at syntactical level, we can verify the absence of ambiguity (for example, two transitions starting from the same state must have different keys),

- at semantic level, input parameter lists for actions must be consistent with the parameters stored into the interactors. The following description (nxt figure) must be added to the description of figure 11:

```
ENTITY state  ;
...
DERIVE params:BAG [0:?] OF parameter =calcule(trans);
INVERSE trans : SET [0:?] OF transitions for s_end;
END_ENTITY ;
```

```
ENTITY transition  ;
...
WHERE
action_coherency :BAG(action.in_list)= s_begin.params+key
;
END_ENTITY ;
```

Figure 13: Coherency between actions and parameters

Each state maintains, in its ‘params’ attribute, the set of parameters that have not been yet concumed. When transitions with action are fired, coherency verifications can be made between the action parameters, the set of parameters in the initial state of the transition, and the token used to fire the transition.

We shall notice that this kind of verifications is rarely stated by authors, because most work do not concern multi-object dialogues.

4.1.2. Global properties

Some global properties may easily be verified using EXPRESS. For example, every consumable token in one interactor has to be producible by the subhierarchy of interactors below itself. In the same way, the opposite property is: every producible token in each interactor must potentially be consumed by the subhierarchy of interactors upper itself.

4.2. Validation

We validated our approach using the EXPRESS tool ECCO Toolkit (Staub and Maier 1992). We built a whole EXPRESS model (bigger in this paper) and wrote an instance generator. The ECCO toolkit has been able to check every rule we mentionned above.

5. Future works

The preliminary results we have presented in this paper demonstrate that verification of interactive dialogue may is possible using an EXPRESS modelisation. Nevertheless, lot of work has to be done... We can verify physical files, but we cannot use these files to generate interactive applications (as we do with ATN files). So, we have either to modify our dialogue compiler to let it accept EXPRESS instances as input, or to create a translator from ATN files to EXPRESS physical files.

In a more general way, we only verify the dialogue. This must be extended to the presentation of the application.

Last, every verification we made is static. No dynamic verification is possible. Enhancing verifications with dynamic aspects would make necessary using another part of EXPRESS definition, the EXPRESS-C procedural language.

References

Bass, L., R. Faneuf, et al. (1992). “A Metamodel for the Runtime Architecture of an Interactive System.” SIGCHI Bulletin **24**(1): 32-37.

Carson, G. S. (1993). “Introduction to the Computer Graphics Reference Model.” Computer Graphics **27**(2): 108-119.

Duce, D. A., R. Van Liere, et al. (1990). “An Approach to Hierarchical Input Devices.” Computer Graphics Forum **9**(1): 15-26.

Duke, D. J. and M. D. Harrison (1993). Towards a Theory of Interactors, Amodeus Esprit Basic Research Project 7040.

EXPRESS (1994). The EXPRESS language reference manual, ISO.

Guittet, L. (1995). Contribution à l'Ingéniérie des Interfaces Homme-Machine - Théorie des Interacteurs et Architecture H4 dans le système NODAOO. LISI/ENSMA. Poitiers, Poitiers**:** 196.

Norman, D. (1986). User Centered System Design, Lawrence Erlbaum Associates.

Palanque, P. (1992). Modélisation par Objets Coopératifs Interactifs d'interfaces homme-machine dirigées par l'utilisateur. PHD Thesis, LIS. Toulouse, Toulouse I**:** 320.

Paterno', F. (1994). “A Theory of User-Interaction Objects.” Journal of Visual Languages and Computing **5**(3): 227-249.

PHIGS (1989). Programmers Hierarchical Interactive Graphics System - Functional Description, ISO.

Pierra, G. (1995). Towards a taxonomy for interactive graphics systems. Eurographics Workshop on Design, Specification, Verification of Interactive Systems, Bonas, Springer-Verlag.

Schenk, D. A. and P. R. Wilson (1994). Information Modeling: The EXPRESS Way, Oxford University Press.

Staub, G. and M. Maier (1992). ECCO Toolkit: an Environment for the Evaluation of EXPRESS Models and the Development of STEP based IT Applications, Universitat Karlsruhe.