Dialogue Validation from Task Analysis

Francis Jambon, Patrick Girard, Yohann Boisdron

Jambon F., Girard P., & Boisdron Y. Dialogue Validation from Task Analysis. *Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'99)*, Eds. D.J. Duke & A. Puerta. Springer-Verlag, *Universidade do Minho, Braga, Portugal*, 2-4 June 1999, pp. 205-224.

Dialogue Validation from Task Analysis

Francis JAMBON, Patrick GIRARD and Yohann BOISDRON

LISI / ENSMA[1], Téléport 2, B.P. 109
F-86960 Futuroscope cedex, France
E-mail: {girard, jambon}@ensma.fr
Web: http://www.lisi.ensma.fr/ihm.html

Keywords: Dialogue Validation, Task Analysis, ARCH Architecture Model, H^4 Architecture Model, Dialogue Component.

Abstract:

Up today, formal methods have mainly been used to allow designers to *verify* that software conforms to its specification. In this article, we propose a *validation* method and a tool to analyse whether the design actually fulfils the original requirements for the system. The principle of our validation method is to generate the complete set of possible user interaction sequences from the task analysis. Then, this set is injected in the Dialogue Controller Component of the application. At last, the Dialogue Controller Component's calls to the Functional Core are intercepted, and compared with the user's goals. Our case study is in the general Computer-Aided Design area, in which systems support a huge number of tasks.

1. Introduction

As it is stated in [Fields, Merriam, & Dearden 1997], in recent past years, formal methods have mainly been used to allow designers to *verify* that software conforms to its specification. By the use of a single notation, many authors intended to demonstrate that some properties of the specification –more precisely, that would have to be in the specification– can be enforced by formal study, by the way of model checking or theorem proving.

Today, the field seems to be mature enough to reach a second step that deals with *validation*. It consists in analysing whether the design actually fulfils the original requirements for the system. In the article above mentioned –[Fields, Merriam, & Dearden 1997]– the authors suggest combining different formalisms which are right for some classes of properties. From descriptive and prescriptive viewpoints, the authors propose using formal methods as communication artefacts in the design process. At the end of their analyses, they conclude with a necessary focus shift from a semantic level to a methodological level.

[1] Laboratory of Applied Computer Science, National School of Engineers in Mechanics and Aerotechnics

Nevertheless, the focus of these approaches is always the design process. Even when requirements are explicitly involved, as for example when task analysis is emphasised, the goal is to provide more formal descriptions of these requirements, and to help analysis and design. This approach might be interesting in domains were tasks are well-defined. Unfortunately, that is not always the case. In the Computer-Aided Design area, systems cannot be described as a global task analysis. The reasons are twice. On the one hand, these systems must support a huge number of possible tasks –for example, some systems contain thousands of primitive functions, whose assembly generates up to a hundred of thousand tasks. On the other hand, "designers" are people whose creativity is a major characteristic. So doing, they cannot be restricted to a finite number of diagrams embedded in constrained tasks.

By the way, validation may be required for systems that cannot be exhaustively described by classical task analysis. Let us give two examples of such validation needs: (1) Choosing an existing system requires the purchaser to evaluate the possibilities of the system against his needs, when basic evaluation of every function of the system is not possible, the definition of scenarios to test on the system may be a solution. An "ad hoc" validation would be a good solution. (2) When a new release is distributed, it is important to know whether the system, despite of its new capabilities, is always capable to do what the users made with the previous release. So, an automated validation would be very interesting.

Our aim in this paper is to describe a method, which includes a task definition grammar and automated tools, in order to provide the designer of a CAD system for a practical validation of use cases. This paper is organised as follows: in Section 2, we explore some related approaches. In Section 3, we describe the context of Computer-Aided Design. In Section 4, we detail what kind of validation we want to achieve. At last, Section 5 focus on our case study.

2. Related Work

Many works in HCI use formal methods to check models of the actual systems. Model checking methods are a good illustration of this point. They are based on the evaluation of logical properties on the state transition system obtained from the evolving variables. Among these techniques, we can find temporal logics, Petri nets and so on. In the area of interactive systems, these methods are assumed to have first been used in formal verification of interactive systems [Campos & Harrison 1997]. For example, [Abowd, Wang, & Monk 1995] verify user interfaces with SMV (Symbolic Model Verifier) using CTL (Computational Tree Logic), while [Paternó & Faconti 1992] uses LOTOS to write interactors specifications, and analyse translated finite state machines using ACTL (Action-based Temporal Logic). [Brun 1997] develops a new temporal logic based on formalism, named XTL (eXtended Temporal Logic), to address interruptions in interactive systems' specification. Model checking is also used by Palanque et al. who model user and system by the way of object-oriented Petri nets –ICO– [Palanque, Bastide, & Sengès 1995]. The weakness of these approaches is that the running system has to be proved to conform to the model. More

recently, [D'Ausbourg 1998] used the Lustre language for the automatic validation of user interface systems. In this case, the formal model is deducted from the UIL[2] description of the interface. Assuming the translator is proved, we can consider that we are really working on the actual system. Nevertheless, in most cases, formal models are only "modelling" the system, with no proof of the equivalence between the model and the system.

Using proof systems have quite similar drawbacks. They are systems where the model is described by variables, operations and invariants. The operations must preserve these invariants and a set of other properties (preconditions and/or post conditions). To ensure the correctness of these specifications, a set of proof obligations are generated and they must be proved. The proof system can achieve some of the proofs automatically. Among these techniques, we can mention Z, based on set theory [Spivey 1988], VDM, based on preconditions and post-conditions calculus [Andrews & Ince 1991], and B, based on the weakest precondition calculus [Abrial 1996 ; Dijkstra 1976]. In the Human-Computer Interaction field VDM and Z have been used to define atomic structures like interactors [Duke & Harrison 1993], and Z and Object-Z are now used more extensively [Hussey & Carrington 1997]. HOL (a Higher Order Logic Theorem Prover) has been used in the verification of User Interface specifications [Bumbulis, et al. 1996]. In these methods, we generally have the same problem of separating specifications and models of the system –on which proofs are conducted– from the actual system –which is only supposed to conform to the models. An exception can be found in [Aït-Ameur, Girard, & Jambon 1998], which uses the refinement methodology to reach the code level: the application itself is proved.

Despite this large number of studies, formal methods are not largely used in HCI. One of the reasons is the *formality gap* [Dix 1991], i.e., the mapping from the requirements of the users and the formalism. In fact, even if we assume that formal models used for reasoning are relevant with final applications, formal methods suffer from a lack of readability and usability for non-specialists. Reading formal models is always hard, and the transition between the informal state of requirements and the formal model is quite difficult. Whenever formal approaches address task analysis [Markopoulos, Johnson, & Rowson 1997], such as ConcurrentTaskTree [Paternò, Mancini, & Meniconi 1997] or [Palanque & Bastide 1995], constructed models are very far from task analysis models such as MAD [Scapin & Pierret-Golbreich 1990] or HTA [Shepherd 1989]. Moreover, while these methods are devoted to analysis and design, no real method or tool is developed to allow system users or experts in human factors to directly validate the systems.

Obviously, a strong need exists for validation techniques, able to ensure that a system "is really able to do that". We describe hereafter a couple of method and tool that allows it in the particular domain of Computer-Aided Design.

[2] User Interface Language

3. Computer-Aided Design Context

Our work has been made in the context of Computer-Aided Design. In this area, systems have specific topics like a huge number of functions, strong relations between functional core and presentation layer, and complex dialogues. In fact, they constitute one of the worse case of applications according to the taxonomy of interactive systems [Pierra 1995]. In this taxonomy, seven criteria have been developed to classify applications' needs. They deals with task arity (1) and structuring (2), domain objects autonomy (3) and structuring (4), source of control (5), mono/multi-user (6) and mono/multi modal (7) applications. The first four criteria are specially relevant for CAD applications. In this section, the terminology conforms to the Arch model [Bass, et al. 1992].

3.1 Task arity

Applications support different kinds of tasks. They can be mono-object tasks –each user task involves only one domain object– or multi-object tasks –each task involves several domain objects. Mono-object tasks may be supported by direct manipulation techniques, whatever multi-object tasks must be represented independently from the domain objects. Some dialogue component, of which the structure is independent from the object structure, must exist. As an example, MacDraw™ only supports mono-object tasks. Examples of multi-object tasks are provided by the drafting systems that enable creating lines as tangential to two circles.

3.2 Task structuring

Applications support atomic tasks when users must specify independently each of their tasks, the result of these tasks are recorded in the state of the domain-specific component. Conversely, applications support structured tasks when users may input in pre-order their task/sub-task hierarchy [Norman 1986]. The support of structured tasks needs recording the state of the dialogue independently from the state of the domain-specific component and the interaction component. Atomic tasks may be encapsulated either in domain objects or in interaction objects.

The following example (fig. 1) shows a typical structured task in which commands are in bold and <> represents conceptual objects that are picked up by the user.

```
create_circle_centre_radius
      projection
               <point_1>
               <line_2>
      distance
               <point_3>
               <point_4>
      /
               2.0
```

Figure 1: A structured task

We call *create_circle_centre_radius* a **terminal task**: it corresponds to one of the goals of the application, and *projection* a **production sub-task**: its role is to *produce* some information token, e.g., a position computed by projecting *<point_1>* on *<line_2>*, for the higher level terminal task.

3.3 Domain objects autonomy

The domain objects are autonomous when their presentation is mainly dependent on their state. In the opposite, they are relational when their presentation depends on the state of other domain objects. If these objects are autonomous, each of them may be mapped onto one interaction object that supports its rendering function. In the other case, rendering spaces must be provided. CAD objects are relational since that they are parts of complex models whose visualisation depends on the view the user selects.

3.4 Domain objects structuring

Domain objects are structured when several levels of objects, structured by aggregation, may be accessed by the user, whenever they are simple when domain objects are not part of other domain objects. When they are highly structured, designing one domain object may only be interpreted by the domain-side components. It requires a complete traversal of the system by the user-defined events.

3.5 Designing a specific model for CAD systems

So, complex interactive graphic applications are systems that support multi-object structured tasks and of which the conceptual objects are structured and relational objects. CAD systems are good examples of such applications. In these systems, conceptual objects are very precise: line are defined by their two end points and sweeps by their sweeped face and sweeping vectors. The tasks are multi-object tasks, users know these constraints and the system must provide for the specification of these constraints [Roller 1990]; the system supports expressions and the tasks are structured.

The conceptual objects are highly structured –a solid is the part inside of a closed shell, a shell consists of faces, etc.– and relational objects –the visibility of a point depends on every entity involved in the hidden surface process. Therefore, it does not exist any one to one mapping between conceptual objects and presentation objects, and when the designer picks up some graphical position, in order to designate some conceptual object, its intent may only be interpreted by querying the functional core.

Such systems are complex software systems, and it is well known, in Software Engineering, that these systems, whether they are designed according to object oriented techniques or not, must be first split into sub-systems. The Arch [UIMS 1992] model provides for such a macro-structuring of the system. But Software Engineering principles also require the interfaces, specification and relationships between these sub-systems, to be precisely defined, and require each system to map to one unique

abstraction. We have developed a specialisation of the Arch model for CAD systems, called the H^4 architecture [Guittet 1995].

The macro-structure of this model (fig. 2) conforms to the Arch model. Regarding its macro-structure, each component, but the domain-adaptor, is based on a specific hierarchy of units. Their role, interface and structuring criteria, have been precisely defined.

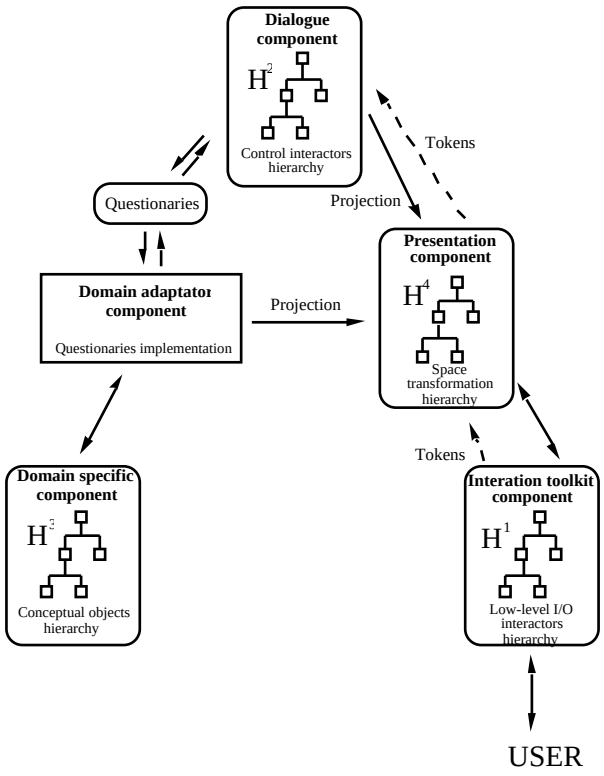

Figure 2: The H^4 architecture model

In our purpose, let us emphasise the dialogue component. The only role of the *dialogue component* is to support the structured and multi-object task-level protocol. When using a multi-agent approach to specify this component, the agents involved in this component are very different from the classical interactive agents such as interactors [Duke & Harrison 1993]. They do not only differ by the abstraction they implement –dialogue component agents implement one task or one category of related tasks such as defining geometrical 3D entities, defining 2D drafting presentation, producing geometric positions from geometric expressions, and so on)– but they also differ by their *interface* with the remaining part of the system. The main role of these agents, that we call *control interactors,* is to control, i.e., to trigger the functional core procedures. The only exchange of events is not sufficient to achieve this goal.

The state of the dialogue does not depend upon the output data that are issued by the functional core as a feedback of the triggered procedures: it only depends on whether or not the procedure fails. Therefore there is no reason to justify that the output flow from the domain adaptor component to the user should go through the dialogue component. This fact was acknowledged by the Seeheim model [Pfaff 1985], and it is still required by the class of applications we discuss in this paper.

Therefore, a control interactor does not provide rendering functions. To ensure the independence between tasks and sub-tasks, a special kind of event dispatcher –called a *monitor*– is needed to realise the circulation of events. It must be noticed that the development of dialogue interactors is completely different from the one of I/O interactors. Intended to support application specific tasks, they cannot be found in any predefined standard toolkit. Fortunately, they may be specified either by using a language approach [Olsen 1992], using an ATN approach [Woods 1970] or Petri Nets [Palanque 1992] in a pure declarative way. Therefore, they may be automatically generated and the different models proposed for dialogue specification [Green 1986] may be used to specify, *in a modular way*, each dialogue interactor behaviour. Do notice that this approach is close to the "transducer" approach from [Accott, et al. 1997]. The major difference consists in the main structure and the global behaviour of our model, which involves elementary "bricks", such as transducers.

Thanks to this architecture, the composition of complex dialogues is straightforward. Organising functions in separate dialogue interactors according to global levels of functions –picking, graphical expressions, geometrical creations, structuring, etc.– allows a good modular decomposition of the application. Then, the hierarchical organisation of the dialogue interactors, under the control of a monitor, allows free compositions of task/sub-tasks provided they do not belong to the same dialogue interactor.

3.6 A strong need for validation

Unfortunately, task analysis is very difficult in CAD systems. Because of the strong hierarchy of goals and sub-goals, which leads to strong hierarchy of tasks and sub-tasks. Task analysis of such systems leads to extremely large and flat trees. Reaching a given goal may be made by several methods with an equivalent result. Defining every possible path is unrealistic. In practice, the definition of CAD systems is made by incremental adjunction of new functions, which are integrated into the dialogue. A design lifecycle starting from a task analysis does not apply.

So, the need for validation is different from systems with well-characterised tasks, as for example air-traffic control or database management. In our approach, we do not address any ergonomic aspect of HCI applications. In the opposite, we focus on dynamic dialogue control. The question is "Is the system able to do *that*", in which "*that*" is expressed in a user-comprehensible way. We choose a notation that is in fact a restriction of MAD [Scapin & Pierret-Golbreich 1990]. This method allows us to define abstract scenarios of dialogues to be validated on the system.

A second requirement is to automate the validation. So, our objective is to produce test sequences from the scenarios' descriptions, and then, to validate the global execution of these test sequences onto the system itself. Test sequences are not an example of system execution, because no real value is provided. The test generation only produces dialogue tokens without associated value. In fact, we are only able to *validate* the dialogue of the interactive system. Let us give an example. Assume we need to test if creating a circle, which centre is the projection of a point on a segment, and which radius is half the distance between a second point and the extremity of the segment, is still possible in a new release (Fig. 3).

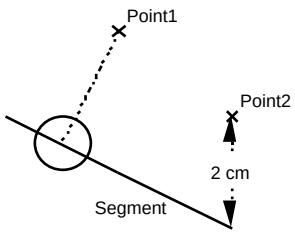

Figure 3: Creation of a circle

For user convenience, we want the system to allow either entering first the centre or the radius of the circle. The task is well-defined: the main goal may be split in two sub-goals, defining the centre and defining the radius, with no sequential relation between them. Each of them must then be expressed by a projection and a distance calculus. Expressing this task in MAD is straightforward. Of course, in real world applications, a huge set of these complex tasks may have to be checked, and so, the validation cannot be hand-performed. From this description, our system must be able to automate the generation of tests sequences and their validation on the system.

At this point, we can list the major requirements for our validation system: the designer's needs, expressed in terms of a goal and sub-goals hierarchy, lead us to use a task analysis as the main data input for our validation system. In addition to this, the designer should want to be sure that the user's goals are really achieved, i.e., correct side effects are the consequences of the user's actions. So, we have to check the system's links to the functional core of the system too. On a more technical point of view, the need for a complete coverage and the huge set of possible user interactions with the target system force us to develop an automated tool to ensure a realistic validation. We will show in the next section our validation principles, and their application on our case study in section 5.

4. Validation Principles

This section deals with the validation principles, whereas the next one (5) is dedicated to the tools used to ensure the validation of our case study –a prototype of a CAD application. In this section, we first detail the HCI properties we want to validate, and then, we describe the validation methodology starting from task analysis to interaction sequence generation.

4.1 Usability Properties

As detailed in the previous section of this paper, our research focuses on CAD systems. These systems usually gather a very important set of functions. In such context, our aim is twofold: we want to make sure that all these functions are reachable for the user, and consistent with the domain objects. So, following Dix and al. [Dix, et al. 1998] in §4.3.3, we have to check both *Reachability* and *Task completeness.*

Reachability

On the one hand, our goal is to check that the system implements the full set of user's tasks given in the user interface specifications. More precisely, we want to make sure that the system accepts and recognises the sequence of user interactions –command selection, object designation, etc.– used to perform each task given in the specifications, and each of its variants. So, from the A. Dix and al. point of view, we want to ensure *Reachability* which "refers to the possibility of navigation through the observable system states".

Reachability is a common usability principle. This principle may be checked by the way of verification of formal specifications. As for example, P. Palanque et al. use an object-oriented Petri nets formalism and a model checking method to prove some Reachabily properties [Palanque, Bastide, & Sengès 1995]. The Y. Aït-Ameur et al. approach is quite different: they use the B method and theorem proving to ensure this principle [Aït-Ameur, Girard, & Jambon 1998].

Our approach in this paper is significantly different: we want to validate the Dialogue Controller of an existing application. We do not provide any model of the final system –in fact we use the binary code of the Dialogue Controller itself. We just need a model of the user tasks performed in an independent way by a specialist of human factors. Another difference is in the scale: we need a complete coverage validation of the system, not only partial proofs.

Task completeness

On the other hand, we want to ensure that the modifications of the domain objects are consistent with the user's tasks. In other words, we must show that the side effects of the user's tasks on the Functional Core can be linked with the user's goal, i.e. the system does exactly what the user wants it to do. So, from the A. Dix and al. point of view, we want to ensure *Task completeness* which "refers to the level to which the system services can be mapped onto all the user tasks".

Task completeness is a usability principle rather difficult to prove, because one has to bridge the gap between user's goals and system Functional Core. In our approach, we want to check that the Functional Core methods called by the Dialogue Controller during user interaction are consistent with the goal and sub-goals defined in the task analysis. This way, we only make the validation on the Dialogue Controller. So, to ensure full Task completeness, we make the assumption that the Functional Core

methods are functionally correct. Such validation of the Functional Core can be carried out separately.

4.2 Task Model

The first step of our validation method is a task analysis, which is supposed to be performed by a specialist in human factors. This task analysis gathers all the possible interactions performed by the user to achieve his goal. And in CAD systems, these user's goals are creations of conceptual objects in the model, modifications or requests for information on these objects.

Typical user's tasks of CAD systems –see §3– are structured. They can be modelled by a classical task/sub-task tree corresponding to a goal/sub-goal hierarchical analysis. In our task model, as in MAD, sub-tasks can be re-used, but cycles and recursivity are forbidden. Moreover, the set of temporal relationships between tasks is restricted to sequence, alternative, and order independence. So our task model is limited to this set of relationships.

We defined a new and simple task model for our kind of applications. This model can be considered as a simplified version of HTA [Shepherd 1989] or MAD [Scapin & Pierret-Golbreich 1990] notations underlying model. The resulting task model will enable us to generate all the possible user's interaction sequences.

4.3 User's Interaction Sequences

It is not straightforward to validate the Dialogue Controller directly from a user's tasks model. However, it is easy to check if a sequence of user's interactions is valid: one just have to do it with the system and see what happened. Our validation method follows a similar principle: we validate the user's interaction sequences one by one.

To do so, we use the task analysis to generate all the possible interaction sequences. These sequences are automatically generated by the way of a classical tree traversal –cycles and recursivity are forbidden from our task model. All the possible sequences are generated, so, we ensure a complete coverage validation of the application. These sequences are finally sent to the system to be validated.

4.4 Dialogue Component Validation

We assume that the target system has been implemented with the Arch architecture model [Bass et al. 1992]. In fact, our case study has been implemented with the H^4 architecture model –see §3.5– which is a variant of the Arch model. Our aim is to validate the Dialogue component of the final application.

As shown on figure 4, the principle of our validation method is to generate the complete set of possible user interaction sequences from the task analysis. Then, this set is injected in the Dialogue Component of the real application via a modified Presentation Component. Then, the application is launched and the Dialogue

Component calls to the Domain Specific Component –the Functional Core– via the Domain Adaptor Component are intercepted, and compared with the user's goal. If they match exactly, the sequence of interactions is assumed to be valid.

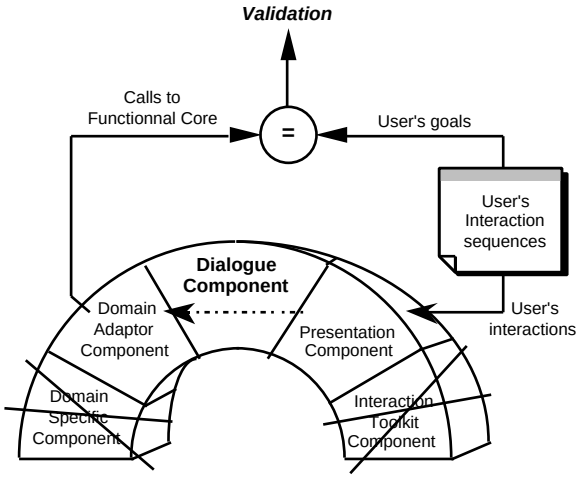

Figure 4: Dialogue component validation principle in the Arch model

This validation principle considers the Dialogue Component as a black box. No assumptions are made about the internal architecture of this component. It must onlyto respect the Application Programming Interface of both the Presentation and the Domain Adaptor Components which is reused by the modified components. So, it is possible to use the binary code of an existing application directly, provided the programming language accept post-compilation linkage.

5. Case study

This section deals with a validation example. We apply the validation principles exposed in the previous section (4) to a case study: MiniCAD, an application of the GIPSE system. In this section, we first describe the main features of MiniCAD, and then we relate the validation process from task analysis to error trace generation.

5.1 Test Application

The GIPSE system is a prototype of a CAD application with an integrated Programming by Example Interface Generator [Patry & Girard 1997]. GIPSE is implemented with the H^4 architecture model described in §3.5. We use a part of this system, called MiniCAD, as our case study. Briefly speaking, the MiniCAD application implements basic CAD functions –objects drawings, objects selections, etc.– and enables structured dialogue modes, but does not include any programming by demonstration generator.

Although MiniCAD is a prototype with very little functions available –compared with a real world CAD system– a huge set of user interactions are possible. As a

consequence, the system cannot be validated “by hand”. So, we develop a tool to do so. We test this tool on the complex structured task previously described: «creating a circle, which centre is the projection of a point on a segment, and which radius is half the distance between a second point and the extremity of the segment» (see fig. 3). For user convenience, the system allows the user either entering first the centre or the radius of the circle. The circle’s radius can also be given in a simpler way by typing directly its value.

5.2 Task Description Grammar

The task of creating a circle can easily be written down by a specialist in human factors thanks to a graphical formalism, like HTA or MAD. We use a simplified version of this latter formalism to express the task of creating a circle. MAD [Scapin & Pierret-Golbreich 1990] is a graphical formalism based on temporal relationships and task/sub-task decomposition. We use as temporal relationships the more relevant operators of an enhanced version of MAD, MAD* [Hammouche 1995]. Among these operators, we use:

- AND which means that the tasks must be executed both in any order ;
- OR which means that only one of the tasks must be executed ;
- SEQ which means that the tasks must be executed both, and in sequence.

The figure 5 shows the resulting specification. In this diagram, the sentences in rectangles are tasks whereas sentences in *Italics* are final atomic tasks directly executed by the user on the interface, as selecting an object by a mouse click or a typing of a number with the keyboard.

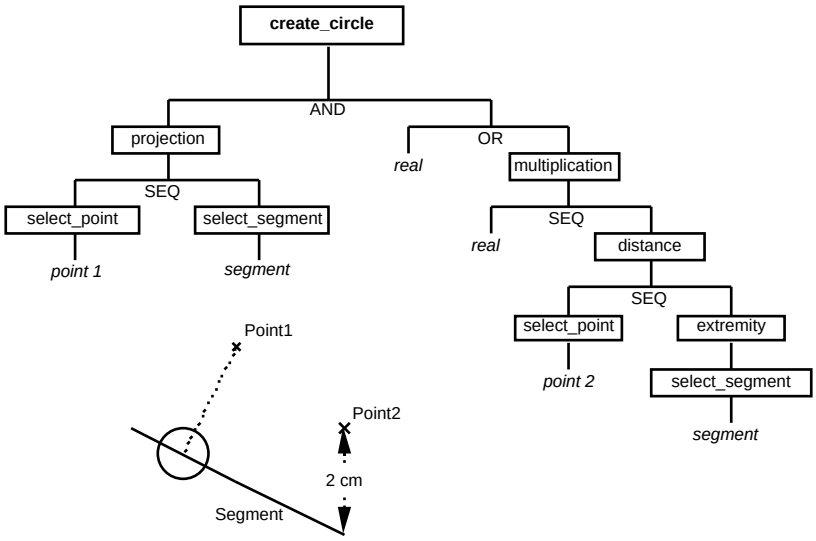

Figure 5: Task of creating a circle in a simplified version of the MAD formalism.

Of course, our sequence generator cannot directly interpret the diagram on figure 5. So, we define a task grammar. Both representations –graphical and text– are equivalent. The tool to translate the graphical representation to the textual one is not yet implemented. Consequently, the task description shown on figure 6 has been translated manually from the MAD description (fig. 5).

```
## task 1

    command : create_circle
    result : circle
    parameter : position=<11>
    & parameter : real=a_real / <13>

    # sub-task 11
              command : projection
              result : position
              parameter : point=<111>
              parameter : segment=<112>
    # end sub-task

    # sub-task 13
              command : multiplication
              result : real
              parameter : real=a_real
              parameter : real=<12>
    # end sub-task

# sub-task 12
              command : distance
              result : real
              parameter : point=<121>
              parameter : position=<122>
    # end sub-task

    # sub-task 122
              command : extremity
              result : position
              parameter: segment=<1221>
    # end sub-task

    # sub-task 111
              command : select_point
              result : point
              parameter: position=a_position
    # end sub-task

    # sub-task 112
              command : select_segment
              result : segment
              parameter : position=a_position
    # end sub-task

    # sub-task 121
              command : select_point
              result : point
              parameter : position=a_position
    # end sub-task

    # sub-task 1221
              command : select_segment
              result : segment
              parameter : position=a_position
    # end sub-task

## end task
```

Figure 6: Task of creating a circle in our task grammar.

In our task grammar, the tasks and sub-tasks are numbered. Each task (resp. sub-task) description begins with the keyword *task* (resp. *sub-task*) and ends with the same keyword with the word *end* before. Then, three types of fields must be fulfilled:

- The *command* field refers to the command activated by the user, for example, the menu's item he must select in order to accomplish his task ; this field is also used to check the Functional Core calls ;
- The *result* field is the type of conceptual object the task or sub-task is supposed to produce, for example, the sub-task extremity (of a segment) command returns a position;
- The *parameter* field gives information about the sub-tasks of the present task:
  - First, it gives the type of the expected conceptual object (*position* or *real* for example);
  - Second, it gives the possible sub-tasks, which can be either the number of a sub-task (*<11>* for example) or an atomic action (*a_position* or *a_real* for example); these sub-tasks are supposed to produce the conceptual object type expected ;
  - Third, a temporal operator can be used to set the relationships among the order of parameters –in fact sub-tasks– or the use of more than one sub-task as a parameter. These operators are equivalent to MAD operators: "&" is equivalent to AND, "/" is equivalent to OR, and the SEQ operator is default.

5.3 Dialogue Sequences Generator

From the task analysis, we can extract the user's interaction sequences. Our task model does not allow neither recursivity nor cycle, so, the task/sub-task analysis specify a finite number of user's interaction sequences. Consequently, we can achieve a complete coverage validation of the Dialogue component of the MiniCAD application.

We develop a tool to extract automatically all these possible interaction sequences. The task analysis (fig. 6) gives four possible interaction sequences. We reprint two of them on figure 7. The syntax of the interaction sequences in similar to the task grammar syntax, except that each sequence begins with the keyword *SEQUENCE* and ends with the keyword *END_SEQUENCE*. When commands are issued by the user (*create_circle* or *select_point* for example) the "`COMMAND` :" sentence is added before the command name. When the user's actions are atomic actions (*a_position* or *a_real* for example), their names are directly printed.

```
SEQUENCE
    COMMAND : create_circle
    a_reel
    COMMAND : projection
    COMMAND : select_point
    a_position
    COMMAND : select_segment
    a_position
END_SEQUENCE
```

```
SEQUENCE

    COMMAND : create_circle
    COMMAND : projection
    COMMAND : select_point
    a_position
    COMMAND : select_segment
    a_position
    COMMAND : multiplication
    a_reel
    COMMAND : distance
    COMMAND : select_point
    a_position
    COMMAND : extremity
    COMMAND : select_segment
    a_position
END_SEQUENCE

…
```

Figure 7: Interaction sequences.

5.4 Test Platform

Our case study is the MiniCAD application (fig. 8). MiniCAD was developed in the ADA95 language with the H^4 architecture model. Our test platform re-uses the Dialogue Component of the MiniCAD application to validate it.

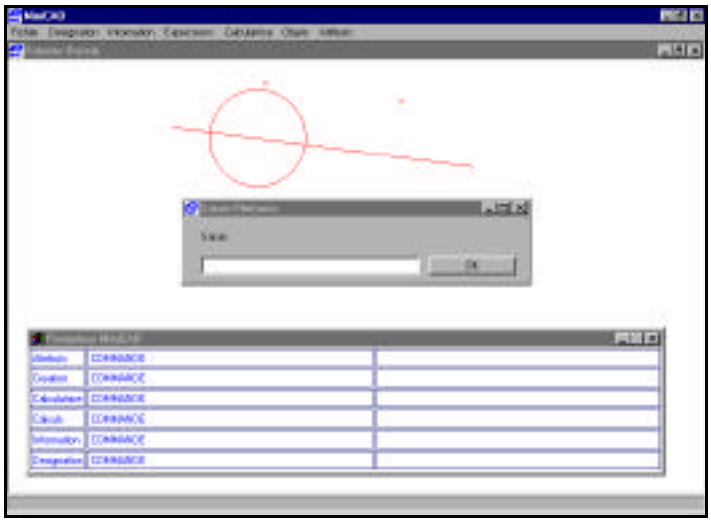

Figure 8: MiniCAD application after the completion of the task “drawing a circle”.

In order to make the mapping between the command names of the task analysis done by the human factors specialist –in English in our case study– and the names of functions implemented by the MiniCAD programmer –in French in our case study–

we need an association table. This table embodies the links between terminal tasks and functions of the CAD system. This table is in fact the only formal link between the human factor domain and the Application Programming Interface (API). The filling of this table needs cooperation between the designers. This table are to be filled only once in the design process, because the task names as well as the API are usually reused for new application versions. It must be update if new tasks are added or API are changed. Figure 9 gives an excerpt of the table we use, in which task analysis command names are on the left side of the "=" symbol, whereas the API methods are on the right one.

```
command : create_circle = COMMANDE : creation.cercle
command : create_segment = COMMANDE : creation.segment
command : create_point = COMANDE : creation.point
command : multiplication = COMMANDE : calc.multiplication
command : projection = COMMANDE : calculs.projection
command : distance = COMMANDE : calculs.distance
command : select_circle = COMMANDE : designation.cercle
command : select_segment = COMMANDE : designation.segment
command : select_point = COMMANDE : designation.point
command : extremity = COMMANDE : information.extremite

…

a_reel : = REEL :
a_position : = POSITION :
a_segment : = SEGMENT :
a_point : = POINT :
a_circle : = CERCLE :

…
```

Figure 9: Excerpt from the association table.

For the test platform, we made a new version of the Presentation component and the Domain Adaptor Component but the Dialogue Component is exactly the same as the one of the MiniCAD application (fig. 10):

- The *Presentation Component* takes the simulated user's interaction sequences and the association table files as inputs, makes the mapping between the sequences and API functions –tokens in the H^4 model– and then, activates the Dialogue Component.
- The *Domain Adaptor Component* has the same API as the MiniCAD application but does not call the Domain Specific Component. This module compares the Dialogue Component calls to the expected commands. When the command called are not the expected ones, or if no command has been called, the test system produces an error for the corresponding sequence. Every error is written in the output file.

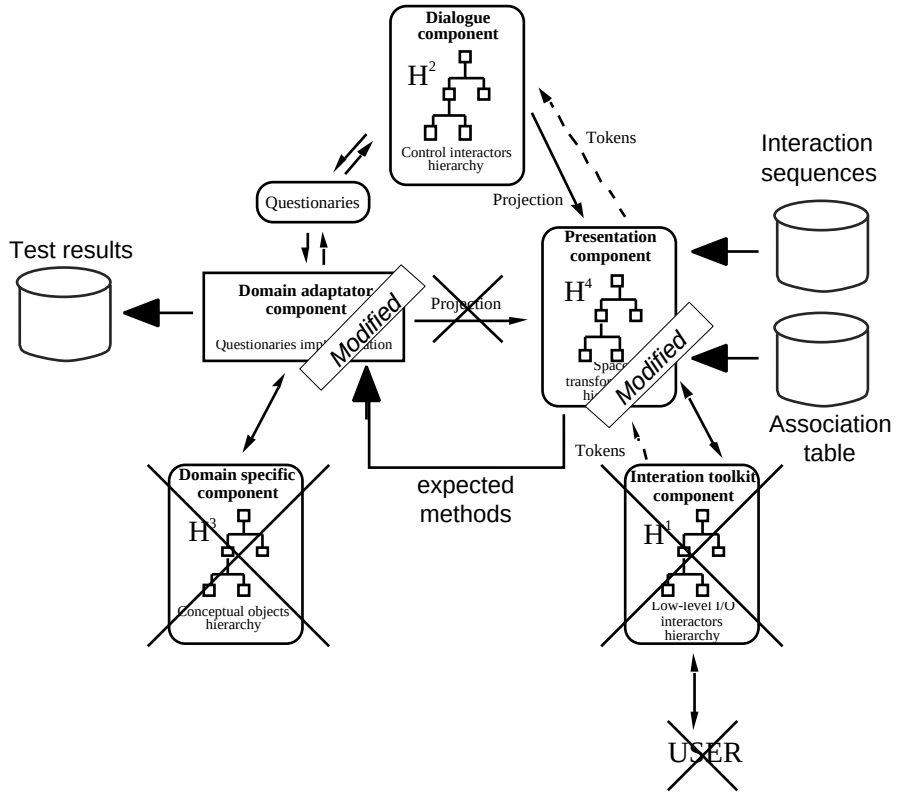

Figure 10: Test platform architecture.

5.5 Execution Traces

We have identified three major validation errors:

- The method called by the Dialogue Component via the Domain Adaptor Component does not exist in the Domain Specific Component API ;
- The method called is not the expected one, e.g., a rectangle is created, whereas a circle is expected.
- No method is called by the Dialogue Component at the end of the sequence.

These errors may be symptoms of either task analysis error or Domain Specific Component bugs. So, they can be rather difficult to interpret by the design team. However, our main goal is to check a huge number of interaction sequences to validate the system as a whole, i.e., to answer the designer “yes all is OK” or “no, there is a problem”.

6. Conclusion

[Fields, Merriam, & Dearden 1997] define focus shifts as new trends for works in DSV-IS papers. In the specific point of validation, our proposal addresses many of them. Shifting from prescription to description, we propose a simple way to describe examples of tasks we want the system to be able to do. Starting from that point, a validation can be done: the analysis performed can determine whether or not the system supports the task. So doing, we also address the need for shifting from verification to validation.

Shifting from general to specific notations allows a great operability. Our model of task description is simple, and allows the automation of the validation process. We are able to perform validation of an existing dialogue component after its design. Moreover, the capabilities of the test platform enable designers to check quickly, while developing a new software version, a huge number of sequences. Doing so, they might prove that a new version is backward compatible with the previous one.

Nevertheless, many points have to be enhanced. The link between terminal tasks and functions of the CAD system is hand-made. With the possible great number of functions (so terminal tasks), a more automated mechanism to establish this link is needed. Methods such as Programming by Demonstration [Cypher 1993] might be used to enforce the usability of our tools. In the same way, a graphical tool might be preferable to pure textual methods. At last, the adaptation of this methodology to other application fields might also be interesting. We also plan to generate interaction sequences not at the Presentation Component level but at a lower level, i.e., at the Interaction Toolkit Component level in order to reduce system code modifications.

7. Bibliography

[Abowd, Wang, & Monk 1995] Abowd G.D., Wang H.-M., & Monk A.F. A Formal Technique for Automated Dialogue Development. *DIS'95, Ann Arbor, Michigan*, August 23-25 1995. p. 219-226.

[Abrial 1996] Abrial J.-R. *The B Book: Assigning Programs to Meanings*. Cambridge University Press, 1996.

[Accott et al. 1997] Accott J., Chatty S., Maury S., & Palanque P. Formal transducers: Models of devices and building bricks for the design of highly interactive systems. *Eurographics Workshop on Design, Specification, Verification of Interactive Systems, Granada, Spain*, June 4-6 1997. p. 143-160.

[Aït-Ameur, Girard, & Jambon 1998] Aït-Ameur Y., Girard P., & Jambon F. Using the B formal approach for incremental specification design of interactive systems. *IFIP Working Conference on Engineering for Human-Computer Interaction (EHCI'98), Heraklion (Crete), Greece*, 14-18 September 1998. p. {to be published}.

[Andrews & Ince 1991] Andrews D. & Ince D. *Practical Formal Methods with VDM*. McGraw-Hill, 1991.

[Bass et al. 1992] Bass L., Faneuf R., Little R., Mayer N., Pellegrino B., Reed S., Seacord R., Sheppard S., & Szczur M.R. A Metamodel for the Runtime Architecture of an Interactive System. *SIGCHI Bulletin*, 1992. vol. 24, n° 1, p. 32-37.

[Brun 1997] Brun P. *XTL: a temporal logic for the formal development of interactive systems*. Formal Methods for Human-Computer Interaction, Springer-Verlag, 1997. p. 121-139.

[Bumbulis et al. 1996] Bumbulis P., Alencar P.S.C., Cowan D.D., & Lucena C.J.P. Validating properties of component-based graphical user interfaces. *Third International Eurographics Workshop on Design, Specification, and Verification of Interactive Systems (DSV-IS'96), Namur, Belgium*, 5-7 June 1996. p. 347-365.

[Campos & Harrison 1997] Campos J.C. & Harrison M.D. Formally Verifying Interactive Systems: A Review. *Eurographics Workshop on Design, Specification, Verification of Interactive Systems, Granada, Spain*, June 4-6 1997. p. 109-124.

[Cypher 1993] Cypher A. *Eager : Programming Repetitive Tasks by Demonstration*. Watch What I Do, Cambridge : The MIT Press, 1993. p. 205-217.

[D'Ausbourg 1998] D'Ausbourg B. Using Model Checking for the Automatic Validation of User Interface Systems. *Eurographics Workshop on Design, Specification and Validation of Interactive Systems (DSV-IS'98), Abingdon, UK*, 1998. p. 242-260.

[Dijkstra 1976] Dijkstra E. *A Discipline of Programming*. Englewood Cliff (NJ), USA : Prentice Hall, 1976.

[Dix et al. 1998] Dix A., Finlay J., Abowd G., & Beale R. *Human-Computer Interaction*. Prentice Hall, 1998.

[Dix 1991] Dix A.J. *Formal Methods for Interactive Systems*. London, UK : Academic Press, 1991.

[Duke & Harrison 1993] Duke D.J. & Harrison M.D. *Towards a Theory of Interactors.* Amodeus Esprit Basic Research Project 7040, 1993 System Modelling/WP6.

[Fields, Merriam, & Dearden 1997] Fields B., Merriam N., & Dearden A. DMVIS: Design, Modelling and Validation of Interactive Systems. *Eurographics Workshop on Design, Specification, Verification of Interactive Systems, Granada, Spain*, June 4-6 1997. p. 29-44.

[Green 1986] Green M.W. *A Survey of three Dialogue Models*. ACM Transactions on Graphics. 1986. vol. 5,n° 3, p. 244-275.

[Guittet 1995] Guittet L. Contribution à l'Ingéniérie des Interfaces Homme-Machine - Théorie des Interacteurs et Architecture H4 dans le système NODAOO. Thèse de Doctorat : Université de Poitiers, Poitiers, 1995.

[Hammouche 1995] Hammouche H. De la modélisation des tâches utilisateurs à la spécification conceptuelle d'interfaces Homme-Machine. PhD : Paris VI, 1995.

[Hussey & Carrington 1997] Hussey A. & Carrington D. *Specifying a Web Browser Interface Using Object-Z*. Formal Methods for Human-Computer Interaction, Springer-Verlag, 1997. p. 157-174.

[Markopoulos, Johnson, & Rowson 1997] Markopoulos P., Johnson P., & Rowson J. Formal Aspects of Task Based Design. *Design, Specification and Verification of Interactive Systems (DSV-IS'97), Granada, Spain*, June 4-6 1997. p. 209-224.

[Norman 1986] Norman D. *User Centered System Design*. Lawrence Erlbaum Associates, 1986.

[Olsen 1992] Olsen D.R. *User Interface Management Systems: Models and Algorithms*. San Mateo (CA), USA : Morgan Kaufmann, 1992.

[Palanque 1992] Palanque P. Modélisation par Objets Coopératifs Interactifs d'interfaces homme-machine dirigées par l'utilisateur. PhD : Toulouse I, Toulouse, 1992.

[Palanque & Bastide 1995] Palanque P. & Bastide R. *Task Models - System Models: a Formal Bridge over the Gap*. Critical Issues in User Interface Engineering, London : Springer-Verlag, 1995. p. 65-80.

[Palanque, Bastide, & Sengès 1995] Palanque P., Bastide R., & Sengès V. Validating interactive system design through the verification of formal task and system models. *IFIP TC2/WG2.7 working conference on engineering for human-computer interaction (EHCI'95), Grand Targhee Resort (Yellowstone Park), USA*, 14-18 August 1995. p. 189-212.

[Paternó & Faconti 1992] Paternó F. & Faconti G.P. *On the LOTOS use to describe graphical interaction*. Cambridge University Press, 1992. p. 155-173.

[Paternò, Mancini, & Meniconi 1997] Paternò F., Mancini C., & Meniconi S. ConcurTaskTrees: A Diagrammatic Notation for Specifying Task Models. *IFIP TC13 human-computer interaction conference (INTERACT'97), Sydney*, 1997. p. 362-369.

[Patry & Girard 1997] Patry G. & Girard P. From Adaptable Interfaces to Model-Based Interface Development: The GIPSE Project. *Third Annual ERCIM Workshop on "User Interfaces for All", Obernai, France*, 3-4 november 1997. p. 127-133.

[Pfaff 1985] Pfaff G.E. User Interface Management Systems, Proceedings of the Workshop on User Interface Management Systems held in Seeheim. Eurographic Seminars. Berlin : Springer-Verlag, 1985.

[Pierra 1995] Pierra G. Towards a taxonomy for interactive graphics systems. *Eurographics Workshop on Design, Specification, Verification of Interactive Systems, Bonas*, June 7-9 1995. p. 362-370.

[Roller 1990] Roller D. *Dimension-Driven Geometry in CAD: a Survey*. Theory and Practice of Geometric Modeling, Springer-Verlag, 1990. p. 509-523.

[Scapin & Pierret-Golbreich 1990] Scapin D.L. & Pierret-Golbreich C. *Towards a method for task description : MAD*. Work with display units 89, Elsevier Science Publishers, North-Holland, 1990.

[Shepherd 1989] Shepherd A. *Analysis and training in information technology tasks*. Task Analysis for Human-Computer Interaction, Chichester, USA : Ellis Horwood, 1989. p. 15-55.

[Spivey 1988] Spivey J.M. *The Z notation: A Reference Manual*. Prentice Hall Int., 1988.

[Woods 1970] Woods W. *Transition Network Grammars for Natural Language Analysis*. Communications of the ACM. 1970. vol. 13,n° 10, p. 591-606.